



INVESTOR IN PEOPLE

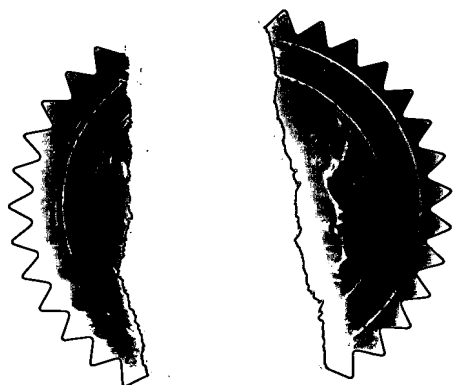
The Patent Office  
Concept House  
Cardiff Road  
Newport  
South Wales  
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.



Signed

Dated 11 November 2003



19NOV02 07:44:09-1 DO  
P01/7700 01:00-0226875

## Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The

Cardif.  
Newpo  
South W  
NP10 8Q

1. Your reference

P015379GB

2. Patent application number

(The Patent Office will fill in this part)

0006870 5

18 NOV 2002

3. Full name, address and postcode of the or of each applicant (underline all surnames)

ARM Limited  
110 Fulbourn Road  
Cherry Hinton  
Cambridge  
CB1 9NJ

Patents ADP number (if you know it)

7498124003

If the applicant is a corporate body, give the country/state of its incorporation

United Kingdom

4. Title of the invention

APPARATUS AND METHOD FOR CONTROLLING ACCESS TO A MEMORY

5. Name of your agent (if you have one)

D Young & Co

"Address for service" in the United Kingdom to which all correspondence should be sent (including the postcode)

21 New Fetter Lane  
London  
EC4A 1DA

Patents ADP number (if you know it)

59006

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and (if you know it) the or each application number

Country

Priority application number  
(if you know it)

Date of filing  
(day / month / year)

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application

Date of filing  
(day / month / year)

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? (Answer 'Yes' if:

Yes

- a) any applicant named in part 3 is not an inventor, or
  - b) there is an inventor who is not named as an applicant, or
  - c) any named applicant is a corporate body.
- See note (d))

APPARATUS AND METHOD FOR CONTROLLING  
ACCESS TO A MEMORY

5 The present invention relates to an apparatus and method for controlling access to a memory.

A data processing apparatus will typically include a processor for running applications loaded onto the data processing apparatus. The processor will operate under the control of an operating system. The data required to run any particular application will typically be stored within a memory of the data processing apparatus. It  
10 will be appreciated that the data may consist of the instructions contained within the application and/or the actual data values used during the execution of those instructions on the processor.

There arise many instances where the data used by at least one of the applications is sensitive data that should not be accessible by other applications that can  
15 be run on the processor. An example would be where the data processing apparatus is a smart card, and one of the applications is a security application which uses sensitive data, such as for example secure keys, to perform validation, authentication, decryption and the like. It is clearly important in such situations to ensure that such sensitive data is kept secure so that it cannot be accessed by other applications that may be loaded onto  
20 the data processing apparatus, for example hacking applications that have been loaded onto the data processing apparatus with the purpose of seeking to access that secure data.

In known systems, it has typically been the job of the operating system developer to ensure that the operating system provides sufficient security to ensure that the secure  
25 data of one application cannot be accessed by other applications running under the control of the operating system. However, as systems become more complex, the general trend is for operating systems to become larger and more complex, and in such situations it becomes increasingly difficult to ensure sufficient security within the operating system itself.

30 Examples of systems seeking to provide secure storage of sensitive data and to provide protection against malicious program code are those described in United States

form of a secure kernel for use when executing certain secure functions. By this approach, the secure domain can be seen as providing a secure world to enable certain sensitive operations to be carried out in a controlled environment. Other applications would then remain in the non-secure domain, which can be considered as a non-secure world.

Furthermore, in accordance with the present invention, the memory comprises secure memory for storing secure data and non-secure memory for storing non-secure data. A memory management unit is provided which is operable, upon receipt of a memory access request issued by the processor when the processor wishes to access an item of data in the memory, to perform one or more predetermined access control functions to control issuance of that memory access request to the memory. As an example, such predetermined access control functions could involve translation of virtual addresses to physical addresses, review of access permissions rights to ensure that the processor in its current mode of operation is entitled to access the data item requested, analysis of region attributes, for example to determine whether the data item is cacheable, bufferable, etc, as will be appreciated by those skilled in the art.

Further, in accordance with the present invention, in order to ensure the security of secure data in the secure memory, partition checking logic is also provided which is managed by the secure operating system, and is operable whenever the memory access request is issued by the processor when it is operating in a non-secure mode, to detect if the memory access request is seeking to access the secure memory. Upon such a detection, the partition checking logic then prevents the access specified by that memory access request from occurring. Since the partition checking logic is managed by the secure operating system, the set-up of the partition checking logic cannot be altered by non-secure applications, thus preventing unauthorised access to the secure data.

The partition checking logic will have access to information about the partitions between the secure memory and the non-secure memory. It will be appreciated that this partitioning information could be hardwired in embodiments where the physical partition between secure memory and non-secure memory cannot be altered. However, in preferred embodiments, the partitioning between secure memory and non-secure memory is configurable by the processor when the processor is operating in a predetermined secure mode, and in such embodiments, the partition checking logic is

the data in the memory. However, it will be appreciated that in alternative embodiments, some degree of partition checking may be provided for certain secure modes of operation.

5 In an alternative embodiment of the present invention, there is at least one particular secure mode of operation in which the memory access request is specified directly by a physical address, and hence in such a particular secure mode, there is no requirement for any virtual to physical address translation to be performed. Whilst the approach of specifying physical addresses directly is less flexible than the virtual address approach, it is inherently more secure since there is no mapping between virtual  
10 addresses and physical addresses to be performed. Hence, in one particularly preferred embodiment, the secure mode that directly specifies physical addresses for memory access requests is the most secure mode of operation, in preferred embodiments this mode being referred to as a monitor mode of operation, and being responsible for managing the transition of the data processing apparatus between the non-secure and the  
15 secure domains.

In such preferred embodiments, the data processing apparatus further comprises a memory protection unit within which the partition checking logic is provided, the memory protection unit being managed by the secure operating system, wherein when the processor is operating in a particular secure mode, the memory access request  
20 specifies a physical address for a memory location, the memory management unit is not used, and the memory protection unit is operable to perform at least memory access permission processing to verify whether the memory location specified by the physical address is accessible in said particular secure mode. Hence, when the processor is operating in a particular secure mode, the access is managed solely by a memory  
25 protection unit managed by the secure operating system.

In preferred embodiments, the memory includes at least one table containing for each of a number of memory regions an associated descriptor, the memory management unit comprising an internal storage unit for storing access control information derived from the descriptors and used by the memory management unit to perform the  
30 predetermined access control functions for the memory access request, the partition checking logic being operable, when the processor is operating in said at least one non-

unit of the memory management unit from storing the altered access control information if an access to secure memory would result, and hence will prevent the access from taking place.

5 The internal storage unit within the memory management unit may take a variety of forms. However, in preferred embodiments the internal storage unit is a translation lookaside buffer (TLB) operable to store for a number of virtual address portions the corresponding physical address portions obtained from corresponding descriptors retrieved from the at least one table.

10 In one embodiment, a single TLB will be contained within the memory management unit, and the partition checking logic will be operable to ensure that any descriptor retrieved from a table in memory will not be stored within the TLB if the processor is operating in non-secure mode and the physical address that would be produced based on the access control information within that descriptor would point to a location in secure memory. When switching between secure and non-secure modes of  
15 operation, the TLB would be flushed to ensure that descriptors relevant to secure mode are not available in non-secure mode, and vice versa.

However, in an alternative embodiment, the internal storage unit consists of both a micro-TLB and main TLB, the main TLB being used to store descriptors retrieved by the memory management unit from the at least one table in memory, and access control  
20 information being transferred from the main TLB to the micro-TLB prior to use of that access control information by the memory management unit to perform the predetermined access control functions for the memory access request. In such embodiments, the partition checking logic is operable, when the processor is operating in said at least one non-secure mode, to prevent the transfer of any access control  
25 information from the main TLB to the micro-TLB that would allow access to said secure memory.

Hence, in such embodiments, descriptors can be copied into the main TLB, but the partition checking logic is operable to police the interface between the main TLB and the micro-TLB when the processor is operating in non-secure mode, to ensure that  
30 no access control information is passed to the micro-TLB that would allow access to the secure memory.

micro-TLB from a descriptor in the main TLB that said associated flag indicates is from the secure table, and in the non-secure mode access control information only being transferred to the micro-TLB from a descriptor in the main TLB that said associated flag indicates is from the non-secure table. The micro-TLB is generally significantly smaller  
5 than the main TLB, and hence the flushing of the micro-TLB whenever the processor moves between the secure domain and the non-secure domain does not have a significant impact on performance. Since the predetermined access control functions performed by the memory management unit are only performed having regard to access control information in the micro-TLB, the above described mechanism ensures that for  
10 any particular mode of operation of the processor, the micro-TLB will only contain access control information derived from descriptors obtained from the appropriate memory table, i.e. from a non-secure table when the processor is operating in a non-secure mode, or a from a secure table when the processor is operating in a secure mode.

In embodiments where all secure modes of operation specify physical addresses  
15 directly within their memory access requests, it will be appreciated that there will not be a need for such a flag within the main TLB, as the main TLB will only store non-secure descriptors.

It will be appreciated that in certain implementations, the memory will not only be accessible by the processor, but may also be accessible by other devices. For  
20 example, in one embodiment, the memory is coupled to a device bus via which a device other than the processor may seek access to the memory. In such embodiments, secure data in the memory needs to be protected not only from rogue applications executed by the processor, but also from any hacking activities initiated via other devices coupled to the device bus. Accordingly, in such embodiments, the data processing apparatus  
25 preferably further comprises additional partition checking logic coupled to the device bus and operable when said device issues a memory access request to said secure memory to only allow access to said secure memory if that device is authorised to access said secure memory.

In preferred embodiments, the additional partition checking logic has regard to a  
30 current mode of operation of the device, and hence for example will only allow the device to have access to a secure part of the memory if that device can verify that it is operating in a secure mode. More particularly, in preferred embodiments, the memory

may cause secure data to be stored in this secure memory part, in which event that secure data will typically be stored in the TCM rather than the external memory, since the TCM will typically have a higher priority. However, if the non-secure operating system subsequently changed again the setting of the physical address range for the

5 TCM so that the previous secure memory part is now mapped in a non-secure physical area of memory, it will be appreciated that the non-secure operating system can then access the secure data, since the partition checking logic will see the area as non-secure and won't assert an abort. Hence, to summarise, if the TCM is configured to act as normal local RAM and not as SmartCache, it may be possible for the non-

10 secure operating system to read secure world data if it can move the TCM base register to non-secure physical address.

In order to avoid this scenario, the data processing apparatus of preferred embodiments is preferably provided with a control flag which is settable by the processor when operating in a privileged secure mode to indicate whether the tightly

15 coupled memory is controllable by the processor only when executing in a privileged secure mode or is controllable by the processor when executing in the at least one non-secure mode. The control flag is set by the secure operating system, and in effect defines whether the TCM is controlled by the secure privileged modes or by non-secure modes. Hence, one configuration that can be defined is that the TCM is only controlled

20 when the processor is operating in a privileged secure mode of operation. In such embodiments, any non-secure access attempted to the TCM control registers will cause an undefined instruction exception to be entered.

In an alternative configuration, the TCM can be controlled by the processor when operating in a non-secure mode of operation. In such embodiments, the TCM is

25 only used by the non-secure applications. No secure data can be stored to or loaded from the TCM. Hence, when a secure access is performed, no look-up is performed within the TCM to see if the address matched the TCM address range. In one preferred embodiment, the TCM is configured so that it is only used by the non-secure operating system, this having the benefit that the non-secure operating system does not

30 need to be changed, since operation proceeds in the normal manner with the TCM available to the non-secure operating system.



Figures 4 and 5 schematically illustrate different relationships between processing modes and security domains;

Figure 6 illustrates one programmer's model of a register bank of a processor  
5 depending upon the processing mode;

Figure 7 illustrates an example of providing separate register banks for a secure domain and a non-secure domain;

10 Figure 8 schematically illustrates a plurality of processing modes with switches between security domains being made via a separate monitor mode;

Figure 9 schematically illustrates a scenario for security domain switching using a mode switching software interrupt instruction;

15

Figure 10 schematically illustrates one example of how non-secure interrupt requests and secure interrupt requests may be processed by the system;

20 Figures 11A and 11B schematically illustrate an example of non-secure interrupt request processing and an example of secure interrupt request processing in accordance with Figure 10;

Figure 12 illustrates an alternative scheme for the handling of non-secure interrupt request signals and secure interrupt request signals compared to that  
25 illustrated in Figure 10;

Figures 13A and 13B illustrate example scenarios for dealing with a non-secure interrupt request and a secure interrupt request in accordance with the scheme  
30 illustrated in Figure 12;

Figure 14 is an example of a vector interrupt table;

Figure 26 is a diagram illustrating the use of both a non-secure page table and a secure page table in preferred embodiments of the present invention;

5        Figure 27 is a diagram illustrating two forms of flag used within the main translation lookaside buffer (TLB) of preferred embodiments;

Figure 28 illustrates how memory may be partitioned after a boot stage in one embodiment of the present invention;

10

Figure 29 illustrates the mapping of the non-secure memory by the memory management unit following the performance of the boot partition in accordance with an embodiment of the present invention;

15        Figure 30 illustrates how the rights of a part of memory can be altered to allow a secure application to share memory with a non-secure application in accordance with an embodiment of the present invention;

Figure 31 illustrates how devices may be connected to the external bus of the data processing apparatus in accordance with one embodiment of the present invention;

20

Figure 32 is a block diagram illustrating how devices may be coupled to the external bus in accordance with the second embodiment of the present invention;

25

Figure 33 schematically shows possible granularity of monitoring functions for different modes and applications running on a processor;

Figure 34 shows possible ways of initiating different monitoring functions;

30

Figure 35 shows a table of control values for controlling availability of different monitoring functions;

defining which activities are to be traced. The trace signals are typically output to a trace buffer from where they can subsequently be analysed. A vectored interrupt controller 21 is provided for managing the servicing of a plurality of interrupts which may be raised by various peripherals ().

5

Further, as shown in Figure 1, another monitoring functionality that can be provided within the core 10 is a debug function, a debugging application external to the data processing apparatus being able to communicate with the core 10 via a Joint Test Access Group (JTAG) controller 18 which is coupled to one or more scan chains 12. Information about the status of various parts of the processor core 10 can be output via the scan chains 12 and the JTAG controller 18 to the external debugging application. An In Circuit Emulator (ICE) 20 is used to store within registers 24 conditions identifying when the debug functions should be started and stopped, and hence for example will be used to store breakpoints, watchpoints, etc.

15

The core 10 is coupled to a system bus 40 via memory management logic 30 which is arranged to manage memory access requests issued by the core 10 for access to locations in memory of the data processing apparatus. Certain parts of the memory may be embodied by memory units connected directly to the system bus 40, for example the Tightly Coupled Memory (TCM) 36, and the cache 38 illustrated in Figure 1. Additional devices may also be provided for accessing such memory, for example a Direct Memory Access (DMA) controller 32. Typically, various control registers 34 will be provided for defining certain control parameters of the various elements of the chip, these control registers also being referred to herein as coprocessor 15 (CP 15) registers.

25

The chip containing the core 10 will typically be coupled to an external bus 70 (for example a bus operating in accordance with the "Advanced Microcontroller Bus Architecture" (AMBA) specification developed by ARM Limited) via an external bus interface 42, and various devices may be connected to the external bus 70. These devices may include master devices such as a digital signal processor (DSP) 50, or a direct memory access (DMA) controller 52, as well as various slave devices such as

30

form a secure operating system. Typically such a secure kernel program 80 will be designed to provide only those functions which are essential to processing activities which must be provided in the secure domain such that the secure kernel 80 can be as small and simple as possible since this will tend to make it more secure. A plurality  
5 of secure applications 82, 84 are illustrated as executing in combination with the secure kernel 80.

Figure 3 illustrates a matrix of processing modes associated with different security domains. In this particular example the processing modes are symmetrical  
10 with respect to the security domain and accordingly Mode 1 and Mode 2 exist in both secure and non-secure forms.

The monitor mode has the highest level of security access in the system and is the only mode entitled to switch the system between the non-secure domain and the  
15 secure domain in either direction. Thus, all domain switches take place via a switch to the monitor mode and the execution of the monitor program 72 within the monitor mode.

Figure 4 schematically illustrates another set of non-secure domain processing  
20 modes 1, 2, 3, 4 and secure domain processing modes a, b, c. In contrast to the symmetric arrangement of Figure 3, Figure 4 shows that some of the processing modes may not be present in one or other of the security domains. The monitor mode 86 is again illustrated as straddling the non-secure domain and the secure domain. The monitor mode 86 can be considered a secure processing mode, since the secure  
25 status flag may be changed in this mode and monitor program 72 in the monitor mode has the ability to itself set the security status flag it effectively provides the ultimate level of security within the system as a whole.

Figure 5 schematically illustrates another arrangement of processing modes  
30 with respect to security domains. In this arrangement both secure and non-secure domains are identified as well as a further domain. This further domain may be such that it is isolated from other parts of a system in a way that it does not need to interact

for the secure domain world may be used, e.g. Figure 6. The monitor mode is responsible switching from one domain to the other. Restoring context, saving previous context, as well as flushing registers is performed by a monitor program at least partially executing in monitor mode. The system behaves thus like a virtualisation model. This type of embodiment is discussed further below. Reference should be made to, for example, the programmer's model of the ARM7 upon which the security features described herein build.

### *Processor Modes*

Instead of duplicating modes in secure world, the same modes support both secure and non-secure domains (see Figure 8). Monitor mode is aware of the current status of the core, either secure or non-secure (e.g. as read from an S bit stored in a coprocessor configuration register).

In the Figure 8, whenever an SMI (Software Monitor Interrupt instruction) occurs, the core enters monitor mode to switch properly from one world to the other.

With reference to Figure 9:

20

1. The scheduler launches thread 1
2. Thread 1 needs to perform a secure function => SMI secure call, the core enters monitor mode. Under hardware control the current PC and CPSR (current processor status register) are stored in R14\_mon and SPSR\_mon (saved processor status register for the monitor mode) and IRQ/FIQ interrupts are disabled.
- 25 3. The monitor program does the following tasks:
  - The S bit is set (the secure status flag).
  - Saves at least R14\_mon and SPSR\_mon in a stack so that non-secure context cannot be lost if an exception occurs whilst the secure application is running.
  - 30 - Checks there is a new thread to launch: secure thread 1. A mechanism (via thread ID table) indicates that thread 1 is active in the secure world.

the associated vector is always fixed and within monitor mode. Its up to the SMI handler to branch to the proper secure function that must be run (e.g. controlled by an operand passed with the instruction).

- 5            Passing parameters from non-secure world to secure world can be performed using the shared registers of the register bank within a Figure 6 type register bank.

When a SMI occurs in non-secure world, the ARM core may do the following actions in hardware:

- 10    - Branch to SMI vector (in secure memory access is allowed since you will now be in monitor mode) into monitor mode
- Save PC into R14\_mon and CPSR into SPSR\_mon
- Set the S bit using the monitor program
- Start to execute secure exception handler in monitor mode (restore/save context in
- 15        case of multi-threads)
- Branch to secure user mode (or another mode, like svc mode) to execute the appropriate function
- IRQ and FIQ are disabled while the core is in monitor mode (latency is increased)

## 20    **Secure World Exit**

There are two possibilities to exit secure world:

- The secure function is finished and we return into previous non-secure mode that had called this function.
- 25    - The secure function is interrupted by a non-secure exception (e.g. IRQ/FIQ/SMI).

## **Normal End of Secure Function**

- The secure function terminates normally and we need to resume an application in the
- 30    non-secure world at the instruction just after the SMI. In the secure user mode, a 'SMI' instruction is performed to return to monitor mode with the appropriate parameters corresponding to a 'return from secure world' routine. At this stage, the registers are

***While in Secure world, if***

- an SIRQ occurs, the core goes to the secure IRQ handler. The core does not leave the secure world
- an IRQ occurs, the core goes to monitor mode where secure context is saved, then to a non-secure IRQ handler to deal with this non-secure interrupt.

In other words, when an interrupt that does not belong to the current world occurs, the core goes directly to monitor mode, otherwise it stays in the current world (see Figure 10).

**IRQ Occurring in Secure World**

See Figure 11A:

1. The scheduler launches thread 1.
2. Thread 1 needs to perform a secure function => SMI secure call, the core enters monitor mode. Current PC and CPSR are stored in R14\_mon and SPSR\_mon, IRQ/FIQ are disabled.
3. The monitor handler (program) does the following tasks:
  - The S bit is set.
  - Saves at least R14\_mon and SPSR\_mon in a stack (and possibly other registers are also pushed) so that non-secure context cannot be lost if an exception occurs whilst the secure application is running.
  - Checks there is a new thread to launch: secure thread 1. A mechanism (via thread ID table) indicates that thread 1 is active in the secure world.
  - Secure application can then start in the secure user mode. IRQ/FIQ are then re-enabled.
4. An IRQ occurs while secure thread 1 is running. The core jumps directly to monitor mode (specific vector) and stores current PC in R14\_mon and CPSR in SPSR\_mon in monitor mode, (IRQ/FIQ are then disabled).

4. The IRQ handler services the SIRQ, then gives control back to the monitor mode handler using an SMI with appropriate parameters.
5. The monitor handler restores non-secure context so that a SUBS instruction makes the core return to the non-secure world and resumes the interrupted thread 1.
- 5 6. Thread 1 executes until the end, then gives the hand back to the scheduler.

The mechanism of Figure 11A has the advantage of providing a deterministic way to enter secure world. However, there are some problems associated with interrupt priority: e.g. while a SIRQ is running in secure interrupt handler, a non-secure IRQ with higher priority may occur. Once the non-secure IRQ is finished, there is a need to recreate the SIRQ event so that the core can resume the secure interrupt.

### **Solution Two**

In this mechanism (See Figure 12) two distinct pins, or only one, may support secure and non-secure interrupts. Having two pins reduces interrupt latency.

15

While in Non Secure world, if

- an IRQ occurs, the core goes to IRQ mode to handle this interrupt like in ARM7 systems
  - a SIRQ occurs, the core goes to an IRQ handler where an SMI instruction will make the core branch to monitor mode to save non-secure context and then to a secure IRQ handler to deal with the secure interrupt.

20

While in a Secure world, if

- a SIRQ occurs, the core goes to the secure IRQ handler. The core does not leave the secure world
  - an IRQ occurs, the core goes to the secure IRQ handler where an SMI instruction will make the core branch to monitor mode (where secure context is saved), then to a non-secure IRQ handler to deal with this non-secure interrupt.

25

### **IRQ Occurring In Secure World**

See Figure 13A:

30



11. The 'return from secure' function does the following tasks:

- indicates that secure thread 1 is finished (i.e., in the case of a thread ID table, remove thread 1 from the table).
- restores from stack non-secure context and flushes required registers, so that no secure information can be read once we return in non-secure world.

- branches back to the non-secure world with a SUBS instruction, restoring the PC (from restored R14\_mon) and CPSR (from SPSR\_mon). The return point in the non-secure world should be the instruction following the previously executed SMI in thread 1.

11. Thread 1 executes until the end, then gives the hand back to the scheduler.

### **SIRQ Occurring in Non-Secure World**

See Figure 13B:

1. The schedule launches thread 1.
2. A SIRQ occurs while secure thread 1 is running.
3. The core jumps directly irq mode and stores current PC in R14\_irq and CPSR in SPSR\_irq. IRQ is then disabled. The IRQ handler detects this is a SIRQ and performs a SMI instruction with appropriate parameters.
4. Once in monitor mode, non-secure context must be saved, then the core goes to a secure IRQ handler.
5. The secure IRQ handler services the SIRQ service routine, then gives control back to monitor with SMI with appropriate parameters.
6. The monitor handler restores non-secure context so that a SUBS instruction makes the core returns to non-secure world and resumes the interrupted IRQ handler.
7. The IRQ handler may then return to the non-secure thread by performing a SUBS.
8. Thread 1 executes until the end, then gives control back to the scheduler.

With the mechanism of Figure 12, there is no need to recreate the SIRQ event in the case of nested interrupts, but there is no guarantee that secure interrupts will be performed.

NB. The Reset entry is only in the secure vector table. When a Reset is performed in non secure world, the core hardware forces entry of supervisor mode and setting of the S bit so that the Reset vector can be accessed in secure memory.

5        Figure 15 illustrates three exception vector tables respectively applicable to a secure mode, a non-secure mode and the monitor mode. These exception vector tables may be programmed with exception vectors in order to match the requirements and characteristics of the secure and non-secure operating systems. Each of the exception vector tables may have an associated vector table base address register  
10        within CP15 storing a base address pointing to that table within memory. When an exception occurs the hardware will reference the vector table base address register corresponding to the current state of the system to determine the base address of the vector table to be used. Alternatively, the different virtual to physical memory mappings applied in the different modes may be used to separate the three different  
15        vector table stored at different physical memory addresses. As illustrated in Figure 16, an exception trap mask register is provided in a system (configuration controlling) coprocessor (CP15) associated with the processor core. This exception trap mask register provides flags associated with respective exception types. These flags indicate whether the hardware should operate to direct processing to either the vector  
20        for the exception concerned within its current domain or should force a switch to the monitor mode (which is a type of secure mode) and then follow the vector in the monitor mode vector table. The exception trap mask register (exception control register) is only writable from the monitor mode. It may be that read access is also prevented to the exception trap mask register when in a non-secure mode. It will be  
25        seen that the exception trap mask register of Figure 16 does not include a flag for the reset vector as the system is configured to always force this to jump to the reset vector in the secure supervisor mode as specified in the secure vector table in order to ensure a secure boot and backwards compatibility. It will be seen that in Figure 15, for the sake of completeness, reset vectors have been shown in the vector tables other than  
30        the secure supervisor mode secure vector table.

within the monitor mode and the monitor program is run at step 102 to handle the CPSR violation exception.

It will be appreciated that the mechanisms for initiating a switch between secure domain and non-secure domain discussed in relation to Figure 17 may be provided in addition to support for the SMI instruction previously discussed. This exception mechanism may be provided to respond to unauthorised attempts to switch mode as all authorised attempts should be made via an SMI instruction. Alternatively, such a mechanism may be legitimate ways to switch between the secure domain and the non-secure domain or may be provided in order to give backwards compatibility with existing code which, for example, might seek to clear the processing status register as part of its normal operation even though it was not truly trying to make an unauthorised attempt to switch between the secure domain and the non-secure domain.

15

A description of an alternative embodiment(s) of the present technique considered from a programmer's model view is given below in relation to Figures 18 to 20 as follows:

20 In the following description, we will use the following terms that must be understood in the context of an ARM processor as designed by ARM Limited, of Cambridge, England.

- S bit : Secure state bit, contained in a dedicated CP15 register.
- 'Secure/Non-Secure state'. This state is defined by the S bit value. It indicates whether the core may access the Secure world (when it is in Secure state, i.e. S=1) or is restricted to the Non-secure world only (S=0). Note that the Monitor mode (see further) overrides the S bit status.
- 'Non-Secure World' groups all hardware/software accessible to non-secure applications that do not require security.
- 30 - 'Secure World' groups all hardware/software (core, memory...) that is only accessible when we execute secure code.
- Monitor mode: new mode that is responsible for switching the core between the Secure and Non-secure state.

Finally, we propose to add some flexibility to the exception handling. All exceptions, apart from the reset, would be handled either in the state where they happened, or would be directed to the Monitor mode. This would be left configurable thanks to a dedicated CP15 register.

The details of this solution are discussed in the following paragraphs.

### **Processor state and modes**

#### **Carbon new features**

#### **Secure or Non-secure state (S bit)**

One major feature of the Carbon core is the presence of the S bit, which indicates whether the core is in a Secure (S=1) or Non-secure (S=0) state. When in Secure state, the core would be able to access any data in the Secure or Non-secure worlds. When in Non-Secure state, the core would be restricted to the Non-secure world only.

The only exception to this rule concerns the Monitor mode, which overrides the S bit information. Even when S=0, the core will perform Secure privileged accesses when it is in Monitor mode. See next paragraph, Monitor mode, for further information

The S bit can only be read and written in Monitor mode. Whatever the S bit value, if any other mode tries to access it, this will be either ignored or result in an Undefined exception.

All exceptions, apart from Reset, have no effect on the Secure state bit. On Reset, the S bit will be set, and the core will start in Supervisor mode. Refer to the boot section for detailed information.

Secure/Nonsecure states are separate and operate independently of the ARM/Thumb/Java states.

In Monitor mode, the MMU will be disabled (flat address map) as well as the MPU or partition checker (the Monitor mode will always perform secure privileged external accesses). However, specially programmed MPU region attributes (cacheability, ...) would still be active.

5

### **New instruction**

This proposal requires adding one new instruction to the existing ARM instruction set.

10       The SMI (Software Monitor Interrupt) instruction would be used to enter the Monitor mode, branching at a fixed SMI exception vector. This instruction would be mainly used to indicate to the Monitor to swap between the Non-secure and Secure State.

15       As an alternative (or in addition) it would be possible to add a new instruction to allow the Monitor mode to save/restore the state of any other mode onto/from the Monitor stack to improve context switching performance.

### **Processor Modes**

20       As discussed in the previous paragraph, only one new mode is added in the core, the Monitor mode. All existing modes remain available, and will exist both in the secure and non-secure states.

In fact, Carbon users will see the structure illustrated in Figure 18.

### **Processor registers**

25       This embodiment proposes that the secure and the non-secure worlds share the same register bank. This implies that, when switching from one world to the other through the Monitor mode, the system monitor will need to save the first world context, and create (or restore) a context in the second world.

30

Passing parameters becomes an easy task: any data contained in a register in the first world will be available in the same register in the second world once the system monitor has switched the S bit.

address. This bit would be programmable by the Monitor or Secure privileged modes only. It would indicate whether the considered interrupt should be treated as Secure, and thus should be handled on the Secure side.

- 5 We would also add two new Vector Address registers, one for all Secure Interrupts happening in Non-Secure state, the other one for all Non-Secure interrupts happening in Secure state.

10 The S bit information contained in CP15 would be also available to the VIC as a new VIC input.

The following table summarizes the different possible scenarios, depending on the status of the incoming interrupt (Secure or Non-secure, indicated by the S bit associated to each interrupt line) and the state of the core (S bit in CP15 = S input signal on the VIC).

15

	Core in secure state (CP15 – S=1)	Core in Non-secure state (CP15 - S=0)
<b>Secure Interrupt</b>	No need to switch between worlds. The VIC directly presents to the core the Secure address associated to the interrupt line. The core simply has to branch at this address where it should find the associated ISR.	The VIC has no Vector associated to this interrupt in the Non-secure domain. It thus presents to the core the address contained in the Vector address register dedicated to all Secure interrupts occurring in Non-secure world. The core, still in Non-secure world, then branches to this address, where it should find an SMI instruction to switch to Secure world. Once in Secure world, it would be able to have access to the correct ISR.

The Reset exception does not have any corresponding bit in this register. Reset will always cause the core to enter the Secure supervisor mode through its dedicated vector.

5

If the bit is set, the corresponding exception makes the core enter the Monitor mode. Otherwise, the exception will be handled in its corresponding handler in the world where it occurred.

10 This register would only be visible in Monitor mode. Any instruction trying to access it in any other mode would be ignored.

This register should be initialized to a system-specific value, depending upon whether the system supports a monitor or not. This functionality could be controlled  
15 by a VIC.

### Exception vectors tables

As there will be separate Secure and Non-secure worlds, we will also need separate Secure and Non-secure exception vectors tables.  
20

Moreover, as the Monitor can also trap some exceptions, we may also need a third exception vectors table dedicated to the Monitor.

The following table summarizes those three different exception vectors tables:  
25

#### In non-secure memory:

Address	Exception	Mode	Automatically accessed when
0x00	-	-	

0x0C	Prefetch Abort	Abort	Aborted instruction when core is in Secure state and Exception Trap Mask reg [Secure PAbort]=0
0x10	Data Abort	Abort	Aborted data when core is in Secure state and Exception Trap Mask reg [Secure Dabort]=0
0x14	Reserved		
0x18	IRQ	IRQ	IRQ pin asserted when core is in Secure state and Exception Trap Mask reg [Secure IRQ]=0
0x1C	FIQ	FIQ	FIQ pin asserted when core is in Secure state and Exception Trap Mask reg [Secure FIQ]=0

\* Refer to “Boot” section for further explanation on the Reset mechanism

**In Monitor memory (flat mapping):**

Address	Exception	Mode	Automatically accessed when
0x00	-	-	-
0x04	Undef	Monitor	Undefined instruction executed when core is in Secure state and Exception Trap Mask reg [Secure Undef]=1 core is in Non-secure state and Exception Trap Mask reg [Non-secure Undef]=1
0x08	SWI	Monitor	SWI instruction executed when core is in Secure state and Exception Trap Mask reg [Secure SWI]=1 core is in Non-secure state and Exception Trap Mask reg [Non-secure SWI]=1



Note that this feature may be limited to a few exceptions, the SMI being one of the most suitable candidates to improve the switching between the Secure and Non-secure states.

## 5      **Switching between worlds**

When switching between states, the Monitor mode must save the context of the first state on its Monitor stack, and restore the second state context from the Monitor stack.

10      The Monitor mode thus needs to have access to any register of any other modes, including the private registers (r14, SPSR, ..).

To handle this, the proposed solution consists in giving any privilege mode in Secure state the rights to directly switch to Monitor mode by simply writing the CPSR.

15

With such a system, switching between worlds is performed as follows:

- enter Monitor mode
- set the S bit
- switch to supervisor mode - save the supervisor registers on the MONITOR stack (of course the supervisor mode will need to have access to the Monitor stack pointer, but this can be easily done, for example by using a common register (R0 to R8))
- switch to System mode - save the registers (=same as the user mode) on the Monitor stack
- IRQ registers on the Monitor stack
- 25 etc ... for all modes
- Once all private registers of all modes are saved, revert to Monitor mode with a simple MSR instruction (= simply write Monitor value in the CPSR mode field)

The other solutions have also been considered:

- 30 - Add a new instruction that would allow the Monitor to save other modes' private registers on its own stack.

3. The secure kernel dispatches the application to the right secure memory location, then switches to user mode (e.g. using a MOVS).
4. The secure function is executed in secure user mode. Once finished, it calls an "exit" function by performing an appropriate SWI.
5. The SWI instruction makes the core enter the secure svc mode through a dedicated SWI vector, that in turn performs the "exit" function. This "exit" function ends with an "SMI" to switch back to monitor mode.
6. The SMI instruction makes the core enter the monitor mode through a dedicated secure SMI vector.
- 10 LR\_mon and SPSR\_mon are used to save the PC and CPSR of the Secure svc mode.  
The S bit remains unchanged (i.e. Secure State).  
The monitor kernel registers the fact that secure thread 1 is finished (removes the secure thread 1 ID from the thread ID table?).
- 15 It then changes the "S" bit by writing into the CP15 register, returning to non-secure state.  
The monitor kernel restores the non-secure context from the monitor stack.  
It also loads the LR\_mon and CPSR\_mon previously saved in step 2.  
Finally, it exits monitor mode with a SUBS, that will make the core return in non-secure user mode, on the instruction
- 20
7. Thread 1 can resume normally.

Figure 21 illustrates in more detail the operation of the memory management logic 30 of one embodiment of the present invention. The memory management logic consists of a Memory Management Unit (MMU) 200 and a Memory Protection Unit (MPU) 220. Any memory access request issued by the core 10 that specifies a virtual address will be passed over path 234 to the MMU 200, the MMU 200 being responsible for performing predetermined access control functions, more particularly for determining the physical address corresponding to that virtual address, and for resolving access permission rights and determining region attributes.

The retrieval of descriptors from the relevant page table into the MMU proceeds as follows. In the event that the memory access request issued by the core 10 specifies a virtual address, a lookup is performed in the micro-TLB 206 which stores for one of a number of virtual address portions the corresponding physical address portions obtained from the relevant page table. Hence, the micro-TLB 206 will compare a certain portion of the virtual address with the corresponding virtual address portion stored within the micro-TLB to determine if there is a match. The portion compared will typically be some predetermined number of most significant bits of the virtual address, the number of bits being dependent on the granularity of the pages within the page table 58. The lookup performed within the micro-TLB 206 will typically be relatively quick, since the micro-TLB 206 will only include a relatively few number of entries, for example eight entries

In the event that there is no match found within the micro-TLB 206, then the memory access request is passed over path 242 to the main TLB 208 which contains a number of descriptors obtained from the page tables. As will be discussed in more detail later, descriptors from both the non-secure page table and the secure page table can co-exist within the main TLB 208, and each entry within the main TLB has a corresponding flag (referred to herein as a domain flag) which is settable to indicate whether the corresponding descriptor in that entry has been obtained from a secure page table or a non-secure page table. In any embodiments where all secure modes of operation specify physical addresses directly within their memory access requests, it will be appreciated that there will not be a need for such a flag within the main TLB, as the main TLB will only store non-secure descriptors.

25

Within the main TLB 208, a similar lookup process is performed to determine whether the relevant portion of the virtual address issued within the memory access request corresponds with any of the virtual address portions associated with descriptors in the main TLB 208 that are relevant to the particular mode of operation. Hence, if the core 10 is operating in non-secure mode, only those descriptors within the main TLB 208 which have been obtained from the non-secure page table will be checked, whereas if the core 10 is operating in secure mode, only the descriptors

within the cache 38, whether in the event of a write access the write data can be buffered, etc.

In the event that there was no hit within the main TLB 208, then the translation table walk logic 210 is used to access the relevant page table 58 in order to retrieve the required descriptor over path 248, and then pass that descriptor over path 246 to the main TLB 208 for storage therein. The base address for both the non-secure page table and the secure page table will be stored within registers of CP15 34, and the current domain in which the processor core 10 is operating, i.e. secure domain or non-secure domain, will also be set within a register of CP15, that domain status register being set by the monitor mode when a transition occurs between the non-secure domain and the secure domain, or vice versa. The content of the domain status register will be referred to herein as the domain bit. Accordingly, if a translation table walk process needs to be performed, the translation table walk logic 210 will know in which domain the core 10 is executing, and accordingly which base address to use to access the relevant table. The virtual address is then used as an offset to the base address in order to access the appropriate entry within the appropriate page table in order to obtain the required descriptor.

Once the descriptor has been retrieved by the translation table walk logic 210, and placed within the main TLB 208, a hit will then be obtained within the main TLB, and the earlier described process will be invoked to retrieve the access control information, and store it within the micro-TLB 206, the access permission logic 202 and the region attribute logic 204. The memory access can then be actioned by the MMU 200.

As mentioned earlier, in preferred embodiments, the main TLB 208 can store descriptors from both the secure page table and the non-secure page table, but the memory access requests are only processed by the MMU 200 once the relevant information is stored within the micro-TLB 206. In preferred embodiments, the transfer of information between the main TLB 208 and the micro-TLB 206 is monitored by the partition checker 222 located within the MPU 220 to ensure that, in

domains, the micro-TLB 206 will be flushed and accordingly the first memory access following a transition between secure domain and non-secure domain will produce a miss in the micro-TLB 206, and require access information to be retrieved from main TLB 208, either directly, or via retrieval of the relevant descriptor from the relevant  
5 page table.

By the above approach, it will be appreciated that the partition checker 222 will ensure that when the core is operating in the non-secure domain, an abort of a memory access will be generated if an attempt is made to retrieve into the micro-TLB  
10 206 access control information that would allow access to secure memory.

If in any modes of operation of the processor core 10, the memory access request is arranged to specify directly a physical address, then in that mode of operation the MMU 200 will be disabled, and the physical address will pass over path  
15 236 into the MPU 220. In a secure mode of operation, the access permission logic 224 and the region attribute logic 226 will perform the necessary access permission and region attribute analysis based on the access permission rights and region attributes identified for the corresponding regions within the partitioning information registers within the CP15 34. If the secure memory location seeking to be accessed is  
20 within a part of secure memory only accessible in a certain mode of operation, for example secure privileged mode, then an access attempt by the core in a different mode of operation, for example a secure user mode, will cause the access permission logic 224 to generate an abort over path 230 to the core in the same way that the access permission logic 202 of the MMU would have produced an abort in such  
25 circumstances. Similarly, the region attribute logic 226 will generate cacheable and bufferable signals in the same way that the region attribute logic 204 of the MMU would have generated such signals for memory access requests specified with virtual addresses. Assuming the access is allowed, the access request will then proceed over path 240 onto the system bus 40, from where it is routed to the appropriate memory  
30 unit.

to Figure 21. The process then proceeds to step 308, or proceeds directly to step 308 from step 304 in the event that the secure descriptor was already in the main TLB 208.

At step 308, it is determined that the main TLB now contains the valid tagged secure descriptor, and accordingly the process proceeds to step 310, where the micro-TLB is loaded with the sub-section of the descriptor that contains the physical address portion. Since the core 10 is currently running in secure mode, there is no need for the partition checker 222 to perform any partition checking function.

10 The process then proceeds to step 312 where the remainder of the memory access proceeds as described earlier.

In the event of a non-secure memory access, the process proceeds from step 300 to step 320, where a lookup process is performed in the micro-TLB 206 to determine whether the corresponding physical address portion from a non-secure descriptor is present. If it is, then the process branches directly to step 336, where the access permission rights are checked by the access permission logic 202. It is important to note at this point that if the relevant physical address portion is within the micro-TLB, it is assumed that there is no security violation, since the partition checker 222 effectively polices the information prior to it being stored within the micro-TLB, such that if the information is within the micro-TLB, it is assumed to be the appropriate non-secure information. Once the access permission has been checked at step 336, the process proceeds to step 338, where it is determined whether there is any violation, in which event an access permission fault abort is issued at step 316. Otherwise, the process proceeds to step 318 where the remainder of the memory access is performed, as discussed earlier.

In the event that at step 320 no hit was located in the micro-TLB, the process proceeds to step 322, where a lookup process is performed in the main TLB 208 to determine whether the relevant non-secure descriptor is present. If not, a page table walk process is performed at step 324 by the translation table walk logic 210 in order to retrieve into the main TLB 208 the necessary non-secure descriptor from the non-

may be no distinction between the two types of abort, whereas in alternative embodiments the abort signal could indicate whether it relates to an access permission fault or a security fault.

- 5           If no violation is detected at step 354, the process proceeds to step 358, where the memory access to the location identified by the physical address occurs.

In preferred embodiments only the monitor mode is arranged to generate physical addresses directly, and accordingly in all other cases the MMU 200 will be  
10   active and generation of the physical address from the virtual address of the memory access request will occur as described earlier.

Figure 22 illustrates an alternative embodiment of the memory management logic in a situation where all memory access requests specify a virtual address, and  
15   accordingly physical addresses are not generated directly in any of the modes of operation. In this scenario, it will be appreciated that a separate MPU 220 is not required, and instead the partition checker 222 can be incorporated within the MMU 200. This change aside, the processing proceeds in exactly the same manner as discussed earlier with reference to Figures 21 and 23.

20

It will be appreciated that various other options are also possible. For example, assuming memory access requests may be issued by both secure and non-secure modes specifying virtual addresses, two MMUs could be provided, one for secure access requests and one for non-secure access requests, i.e. MPU 220 in Figure 21  
25   could be replaced by a complete MMU. In such cases, the use of flags with the main TLB of each MMU to define whether descriptors are secure or non-secure would not be needed, as one MMU would store non-secure descriptors in its main TLB, and the other MMU would store secure descriptors in its main TLB. Of course, the partition checker would still be required to check whether an access to secure memory is being  
30   attempted whilst the core is in the non-secure domain.

memory, the partition checker 222 prevents the access taking place. In contrast, if the physical address that would be derived using this descriptor corresponds to a non-secure memory region, for example region 374 as illustrated in Figure 25, then the access control information loaded into the micro-TLB 206 merely identifies this non-secure region 374. Hence, accesses within that non-secure memory region 374 can occur but no accesses into any of the secure regions 376, 378 or 380 can occur. Thus, it can be seen that even though the main TLB 208 may contain descriptors from the non-secure page table that have been tampered with, the micro-TLB will only contain physical address portions that will enable access to non-secure memory regions.

10

As described earlier, in embodiments where both non-secure modes and secure modes may generate memory access requests specifying virtual addresses, then the memory preferably comprises both a non-secure page table within non-secure memory, and a secure page table within secure memory. When in non-secure mode, the non-secure page table will be referenced by the translation table walk logic 210, whereas when in secure mode, the secure page table will be referenced by the translation table walk logic 210. Figure 26 illustrates these two page tables. As shown in Figure 26, the non-secure memory 390, which may for example be within external memory 56 of Figure 1, includes within it a non-secure page table 395 specified in a CP15 register 34 by reference to a base address 397. Similarly, within secure memory 400, which again may be within the external memory 56 of Figure 1, a corresponding secure page table 405 is provided which is specified within a duplicate CP15 register 34 by a secure page table base address 407. Each descriptor within the non-secure page table 395 will point to a corresponding non-secure page in non-secure memory 390, whereas each descriptor within the secure page table 405 will define a corresponding secure page in the secure memory 400. In addition, as will be described in more detail later, it is possible for certain areas of memory to be shared memory regions 410, which are accessible by both non-secure modes and secure modes.

30

Figure 27 illustrates in more detail the lookup process performed within the main TLB 208 in accordance with preferred embodiments. As mentioned earlier, the



techniques may be used to determine which descriptor to evict from the main TLB to make room for the new descriptor, for example least recently used approaches, etc.

It will be appreciated that the secure kernel used in secure modes of operation  
5 may be developed entirely separately to the non-secure operating system. However, in certain cases the secure kernel and the non-secure operating system development may be closely linked, and in such situations it may be appropriate to allow secure applications to use the non-secure descriptors. Indeed, this will allow the secure applications to have direct access to non-secure data (for sharing) by knowing only the  
10 virtual address. This of course presumes that the secure virtual mapping and the non-secure virtual mapping are exclusive for a particular ASID. In such scenarios, the tag introduced previously (i.e. the domain flag) to distinguish between secure and non-secure descriptors will not be needed. The lookup in the TLB is instead then performed with all of descriptors available.

15

In preferred embodiments, the choice between this configuration of the main TLB, and the earlier described configuration with separate secure and non-secure descriptors, can be set by a particular bit provided within the CP15 control registers. In preferred embodiments, this bit would only be set by the secure kernel.

20

In embodiments where the secure application were directly allowed to use a non-secure virtual address, it would be possible to make a non-secure stack pointer available from the secure domain. This can be done by copying a non-secure register value identifying the non-secure stack pointer into a dedicated register within the  
25 CP15 registers 34. This will then enable the non-secure application to pass parameters via the stack according to a scheme understood by the secure application.

As described earlier, the memory may be partitioned into non-secure and secure parts, and this partitioning is controlled by the secure kernel using the CP15  
30 registers 34 dedicated to the partition checker 222. The basic partitioning approach is based on region access permissions as definable in typical MPU devices. Accordingly, the memory is divided into regions, and each region is preferably

When the partition of the memory is changed, the micro-TLB 206 needs to be flushed. Hence, in this scenario, when a non-secure access subsequently occurs, a miss will occur in the micro-TLB 206, and accordingly a new descriptor will be loaded from the main TLB 208. This new descriptor will subsequently be checked by  
5 the partition checker 222 of the MPU as it is attempted to retrieve it into the micro-TLB 206, and so will be consistent with the new partition of the memory.

In preferred embodiments, the cache 38 is virtual-indexed and physical-tagged. Accordingly, when an access is performed in the cache 38, a lookup will have already  
10 been performed in the micro-TLB 206 first, and accordingly access permissions, especially secure and non-secure permissions, will have been checked. Accordingly, secure data cannot be stored in the cache 38 by non-secure applications. Access to the cache 38 is under the control of the partition checking performed by the partition checker 222, and accordingly no access to secure data can be performed in non-secure  
15 mode.

However, one problem that could occur would be for an application in the non-secure domain to be able to use the cache operations register to invalidate, clean, or flush the cache. It needs to be ensured that such operations could not affect the  
20 security of the system. For example, if the non-secure operating system were to invalidate the cache 38 without cleaning it, any secure dirty data must be written to the external memory before being replaced. Preferably, secure data is tagged in the cache, and accordingly can be dealt with differently if desired.

25 In preferred embodiments, if an "invalidate line by address" operation is executed by a non-secure program, the physical address is checked by the partition checker 222, and if the cache line is a secure cache line, the operation becomes a "clean and invalidate" operation, thereby ensuring that the security of the system is maintained. Further, in preferred embodiments, all "invalidate line by index"  
30 operations that are executed by a non-secure program become "clean and invalidate by index" operations. Similarly, all "invalidate all" operations executed by a non-secure program become "clean and invalidate all" operations.

modes of operation, and provides two possible configurations. In a first configuration, this control bit is set to "1", in which event the TCM can only be controlled by the secure privilege modes. Hence, any non-secure access attempted to the TCM control registers within the CP15 34 will cause an undefined instruction exception to be entered. Thus, in this first configuration, both secure modes and non-secure modes can use the TCM, but the TCM is controlled only by the secure privilege mode. In the second configuration, the control bit is set to "0", in which event the TCM can be controlled by the non-secure operating system. In this case, the TCM is only used by the non-secure applications. No secure data can be stored to or loaded from the TCM. Hence, when a secure access is performed, no look-up is performed within the TCM to see if the address matched the TCM address range.

By default, it is envisaged that the TCM would be used only by non-secure operating systems, as in this scenario the non-secure operating system would not need to be changed.

As mentioned earlier, in addition to the provision of the partition checker 222 within the MPU 220, preferred embodiments of the present invention also provide an analogous partition checking block coupled to the external bus 70, this additional partition checker being used to police accesses to memory by other master devices, for example the digital signal processor (DSP) 50, the DMA controller 52 coupled directly to the external bus, the DMA controller 32 connectable to the external bus via the external bus interface 42, etc. As mentioned earlier, the entire memory system can consist of several memory units, and a variety of these may exist on the external bus 70, for example the external memory 56, boot ROM 44, or indeed buffers or registers 48, 62, 66 within peripheral devices such as the screen driver 46, I/O interface 60, key storage unit 64, etc. Furthermore, different parts of the memory system may need to be defined as secure memory, for example it may be desired that the key buffer 66 within the key storage unit 64 should be treated as secure memory. If an access to such secure memory were to be attempted by a device coupled to the external bus,

For the sake of illustration, the arbiter block 476 used to arbitrate between memory access requests issued by master devices, such as devices 470, 472, is illustrated separately to the decoder 478 used to determine the appropriate memory device to service the memory access request, and separate from the partition checker 492. However, it will be appreciated that one or more of these components may be integrated within the same unit if desired.

Figure 32 illustrates an alternative embodiment, in which a partition checker 492 is not provided, and instead each memory device 474, 480, 484 is arranged to police its own memory access dependent on the value of the S bit. Accordingly, if device 470 were to assert a memory access request in non-secure mode to a register 482 within the screen driver 480 that was marked as secure memory, then the screen driver 480 would determine that the S bit was not asserted, and would not process the memory access request. Accordingly, it is envisaged that with appropriate design of the various memory devices, it may be possible to avoid the need for a partition checker 492 to be provided separately on the external bus.

Figure 33 shows different modes and applications running on a processor. The dashed lines indicate how different modes and/or applications can be separated and isolated from one another during monitoring of the processor according to an embodiment of the present invention.

The ability to monitor a processor to locate possible faults and discover why an application is not performing as expected is extremely useful and many processors provide such functions. The monitoring can be performed in a variety of ways including debug and trace functions.

In the processor according to the present technique debug can operate in several modes including halt debug mode and monitor debug mode. These modes are intrusive and cause the program running at the time to be stopped. In halt debug mode, when a breakpoint or watchpoint occurs, the core is stopped and isolated from the rest of the system and the core enters debug state. On entry the core is halted, the

The processor of embodiments of the present technique operates in two separate domains, in the embodiments described these domains comprise secure and non-secure domains. However, for the purposes of the monitoring functions, it will be clear to the skilled person that these domains can be any two domains between which data should not leak. Embodiments of the present technique are concerned with preventing leakage of data between the two domains and monitoring functions such as debug and trace which are conventionally allowed access to the whole system are a potential source of data leakage between the domains.

In the example given above of a secure and non-secure domain or world, secure data must not be available to the non-secure world. Furthermore, if debug is permitted, in secure world, it may be advantageous for some of the data within secure world to be restricted or hidden. The hashed lines in Figure 33 shows some examples of possible ways to segment data access and provide different levels of granularity. In Figure 33, monitor mode is shown by block 500 and is the most secure of all the modes and controls switching between secure and non-secure worlds. Below monitor mode 500 there is a supervisor mode, this comprises secure supervisor mode 510 and non-secure supervisor mode 520. Then there is non-secure user mode having applications 522 and 524 and secure user mode with applications 512, 514 and 516. The monitoring modes (debug and trace) can be controlled to only monitor non-secure mode (to the left of hashed line 501). Alternatively the non-secure domain or world and the secure user mode may be allowed to be monitored (left of 501 and the portion right of 501 that lies below 502). In a further embodiment the non-secure world and certain applications running in the secure user domain may be allowed, in this case further segmentation by hashed lines 503 occurs. Such divisions help prevent leakage of secure data between different users who may be running the different applications. In some controlled cases monitoring of the entire system may be allowed. According to the granularity required the following parts of the core need to have their access controlled during monitoring functions.

There are four registers that can be set on a Debug event; the instruction Fault Status Register (IFSR), Data Fault Status Register (DFSR), Fault Address Register

Access of the monitoring functions to portions of the core can be controlled at initiation of the monitoring function. Debug and trace are initialised in a variety of ways. Embodiments of the present technique control the access of the monitoring function to certain secure portions of the core by only allowing initialisation under  
5 certain conditions.

Embodiments of the present technique seek to restrict entry into monitoring functions with the following granularity:

- By controlling separately intrusive and observable (trace) debug;
- 10 By allowing debug entry in secure user mode only or in the whole secure world;
- By allowing debug in secure user mode only and moreover taking account of the thread ID (application running).

15 In order to control the initiation of a monitoring function it is important to be aware of how the functions can be initiated. Figure 34 shows a table illustrating the possible ways of initiating a monitoring function, the type of monitoring function that is initiated and the way that such an initiation instruction can be programmed.

20 Generally, these monitoring instructions can be entered via software or via hardware, i.e. via the JTAG controller. In order to control the initiation of monitoring functions, control values are used. These comprise enable bits which are condition dependent and thus, if a particular condition is present, monitoring is only allowed to start if the enable bit is set. These bits are stored on a secure register CP14 (debug and  
25 status control register, DSCR), which is located in ICE 530 (see Figure 41).

In a preferred embodiment there are four bits that enable/disable intrusive and observable debug, these comprise a secure debug enable bit, a secure trace enable bit, a secure user-mode enable bit and a secure thread aware enable bit. These control  
30 values serve to provide a degree of controllable granularity for the monitoring function and as such can help stop leakage of data from a particular domain. Figure 35 provides a summary of these bits and how they can be accessed.

In functional mode, SE (Scan Enable) is clear and the register cell works as a single register cell. In test or debug mode, SE is set and input data can come from SI input (Scan In) instead of D input.

## 5 **Scan chain**

All scan chain cells are chained in scan chain as shown in figure 38.

In functional mode, SE is clear and all register cells can be accessed normally and interact with other logic of the circuit. In Test or Debug mode, SE is set and all registers are chained between each other in a scan chain. Data can come from the first scan chain cell and can be shifted through any other scan chain cell, at the cadence of each clock cycle. Data can be shifted out also to see the contents of the registers.

15

## **TAP controller**

A debug TAP controller is used to handle several scan chains. The TAP controller can select a particular scan chain: it connects “Scan In” and “Scan Out” signals to that particular scan-chain. Then data can be scanned into the chain, shifted, or scanned out. The TAP controller is controlled externally by a JTAG port interface. Figure 39 schematically illustrates a TAP controller

## 25 **JTAG Selective Disable Scan Chain Cell**

For security reasons, some registers might not be accessible by scan chain, even in debug or test mode. A new input called JADI (JTAG Access Disable) can allow removal dynamically or statically of a scan chain cell from a whole scan chain, without modifying the scan chain structure in the integrated circuit. Figures 40A and B schematically show this input.

It should be noted that by default debug (intrusive and observable – trace) are only available in non-secure world. To enable them to be available in secure world the control value enable bits need to be set.

5       The advantages of this are that debug can always be initiated by users to run in non-secure world. Thus, although access to secure world is not generally available to users in debug this may not be a problem in many cases because access to this world is limited and secure world has been fully verified at board stage prior to being made available. It is therefore foreseen that in many cases debugging of the secure world  
10 will not be necessary. A secure supervisor can still initiate debug via the software route of writing CP14 if necessary.

Figure 42 schematically shows the control of debug initialisation. In this figure a portion of the core 600 comprises a storage element 601 (which may be a  
15 CP15 register as previously discussed) in which is stored a secure status bit S indicative of whether the system is in secure world or not. Core 600 also comprises a register 602 comprising bits indicative of the mode that the processor is running in, for example user mode, and a register 603 providing a context identifier that identifies the application or thread that is currently running on the core.

20       When a breakpoint is reached comparator 610, which compares a breakpoint stored on register 611 with the address of the core stored in register 612, sends a signal to control logic 620. Control logic 620 looks at the secure state S, the mode 602 and the thread (context identifier) 603 and compares it with the control values and  
25 condition indicators stored on register CP14. If the system is not operating in secure world, then a “enter debug” signal will be output at 630. If however, the system is operating in secure world, the control logic 620 will look at the mode 602, and if it is in user mode will check to see if user mode enable and debug enable bits are set. If they are then debug will be initialised provided that a thread-aware bit has not been  
30 initialised. The above illustrates the hierarchical nature of the control values.



It is the same for debug granularity concerning observable debug. Figure 43B shows a summary of secure debug granularity in this case, here default values at reset are also represented in grey colour.

- 5           Note that Secure user-mode debug enable bit and Secure thread-aware debug enable bit are commonly used for intrusive and observable debug.

A thread aware initialisation bit is stored in register CP14 and indicates if granularity by application is required. If the thread aware bit has been initialised, the control logic will further check that the application identifier or thread 603 is one indicated in the thread aware control value, if it is, then debug will be initialised. If either of the user mode or debug enable bits are not set or the thread aware bit is set and the application running is not one indicated in the thread aware control value, then the breakpoint will be ignored and the core will continue doing what it was doing and debug will not be initialised.

In addition to controlling initialisation of monitoring functions, the capture of diagnostic data during a monitor function can also be controlled in a similar way. In order to do this the core must continue to consider both the control values, i.e. the enable bits stored in register CP14 and the conditions to which they relate during operation of the monitoring function.

Figure 44 shows schematically granularity of a monitoring function while it is running. In this case region A relates to a region in which it is permissible to capture diagnostic data and region B relates to region in which control values stored in CP14 indicate that it is not possible to capture diagnostic data.

Thus, when debug is running and a program is operating in region A, diagnostic data is output in a step-by-step fashion during debug. When operation switches to Region B, where the capture of diagnostic data is not allowed, debug no longer proceeds in a step by step fashion, rather it proceeds atomically and no data is captured. This continues until operation of the program re-enters region A whereupon

Debug halt mode can be entered from whatever mode and from whatever domain. Breakpoints and watchpoints can be set in secure or in non-secure memory. In debug state, it is possible to enter secure world by simply changing the S bit via an MCR instruction.

5

As debug mode can be entered when secure exceptions occur, the vector trap register is extended with new bits which are;

SMI vector trapping enable

Secure data abort vector trapping enable

10     Secure prefetch abort vector trapping enable

Secure undefined vector trapping enable.

In monitor debug mode, if we allow debug everywhere, even when an SMI is called in non-secure world, it is possible to enter secure world in step-by-step debug.

15     When a breakpoint occurs in secure domain, the secure abort handler is operable to dump secure register bank and secure memory.

20     The two abort handlers in secure and in non-secure world give their information to the debugger application so that debugger window (on the associated debug controlling PC) can show the register state in both secure and non-secure worlds.

25     Figure 45A shows what happens when the core is configured in monitor debug mode and debug is enabled in secure world. Figure 45B shows what happens when the core is configured in monitor debug mode and the debug is disabled in secure world. This later process will be described below.

### **Intrusive debug at production stage**

30     In production stage when JSDAEN is tied and debug is restricted to non-secure world, unless the secure supervisor determines otherwise, then the table shown in Figure 45B shows what happens. In this case SMI should always be considered as an atomic instruction, so that secure functions are always finished before entering debug state.

Before entering monitor mode:

Breakpoints and watchpoints are only taken into account in non-secure world. If bit S is set, breakpoints/watchpoints are bypassed. Note that watchpoints units are also  
5 accessible with MCR/MRC (CP14) which is not a security issue as  
breakpoint/watchpoint has no effect in secure memory.

BKPT are normally used to replace the instruction on which breakpoint is set. This  
supposes to overwrite this instruction in memory by BKPT instruction, which will be  
10 possible only in non-secure mode.

Vector Trap Register concerns non-secure exceptions only. All extended trapping  
enable bits explained before have no effect. Data abort and Pre-fetch abort enable bits  
should be disabled to avoid the processor being forced in to an unrecoverable state.  
15 Via JTAG, we have the same restrictions as for halt mode (S bit cannot be  
modified, etc)

Once in monitor mode (non-secure abort mode)

The non-secure abort handler can dump non-secure world and has no visibility on  
20 secure banked registers as well as secure memory.

Executes secure functions with atomic SMI instruction

S bit cannot be changed to force secure world entry.

Mode bits can not be changed if debug is permitted in secure supervisor mode only.

25 Note that if an external debug request (EDBGRQ) occurs,

In non-secure world, the core terminates the current instruction and enters then  
immediately debug state (in halt mode).

In secure world, the core terminates the current function and enters the Debug State  
when it has returned in non-secure world.

- Reset bit: as we enter secure world when reset occurs, this bit is valid only when debug in secure world is enabled, otherwise it has no effect.

Although a particular embodiment of the invention has been described herein,  
5 it will be apparent that the invention is not limited thereto, and that many  
modifications and additions may be made within the scope of the invention. For  
example, various combinations of the features of the following dependent could be  
made with the features of the independent claims without departing from the scope of  
the present invention.

3. A data processing apparatus as claimed in Claim 1 or Claim 2, wherein when the processor is operating in one of the at least one secure modes, the memory access request specifies a virtual address, the memory management unit is controlled by the secure operating system and one of said predetermined access control functions performed by the memory management unit comprises conversion of the virtual address to a physical address, the partition checking logic not being used in the at least one secure mode.
4. A data processing apparatus as claimed in Claim 3, wherein for all modes of operation of the processor, the memory access request specifies a virtual address, the partition checking logic being provided within the memory management unit, and being operable whenever the processor is operating in said at least one non-secure mode.
5. A data processing apparatus as claimed in Claim 2 or Claim 3, further comprising a memory protection unit within which the partition checking logic is provided, the memory protection unit being managed by the secure operating system, wherein when the processor is operating in a particular secure mode, the memory access request specifies a physical address for a memory location, the memory management unit is not used, and the memory protection unit is operable to perform at least memory access permission processing to verify whether the memory location specified by the physical address is accessible in said particular secure mode.
6. A data processing apparatus as claimed in any preceding claim, wherein the memory includes at least one table containing for each of a number of memory regions an associated descriptor, the memory management unit comprising an internal storage unit for storing access control information derived from the descriptors and used by the memory management unit to perform the predetermined access control functions for the memory access request, the partition checking logic being operable, when the processor is operating in said at least one non-secure mode, to prevent the internal storage unit from storing access control information that would allow access to said secure memory.

information the physical address portion specified by that descriptor if the physical address that would then be produced for the virtual address is within the secure memory.

11. A data processing apparatus as claimed in Claim 10 when dependent on Claim 9, wherein the at least one table further comprises a secure table within the secure memory that contains descriptors generated by the secure operating system, the main TLB comprising a flag associated with each descriptor stored within the main TLB to identify whether that descriptor is from said non-secure table or said secure table.

12. A data processing apparatus as claimed in Claim 11, wherein the micro-TLB is flushed whenever the mode of operation of the processor changes between a secure mode and a non-secure mode, in the secure mode access control information only being transferred to the micro-TLB from a descriptor in the main TLB that said associated flag indicates is from the secure table, and in the non-secure mode access control information only being transferred to the micro-TLB from a descriptor in the main TLB that said associated flag indicates is from the non-secure table.

13. A data processing apparatus as claimed in any preceding claim when dependent on Claim 6, wherein said at least one table comprises at least one page table.

20

14. A data processing apparatus as claimed in any preceding claim, wherein the memory is coupled to a device bus via which a device other than the processor may seek access to the memory, the apparatus further comprising additional partition checking logic coupled to the device bus and operable when said device issues a memory access request to said secure memory to only allow access to said secure memory if that device is authorised to access said secure memory.

25

15. A data processing apparatus as claimed in Claim 14, wherein the additional partition checking logic has regard to a current mode of operation of said device.

30

- (ii) performing one or more predetermined access control functions to control issuance of the memory access request to the memory;
- (iii) whenever the memory access request is issued by the processor when operating in said at least one non-secure mode, employing partition checking logic managed by the secure operating system to detect if the memory access request is seeking to access the secure memory; and
- (iv) upon such detection, preventing the access specified by that memory access request.

21. A method as claimed in Claim 20, wherein when the processor is operating in the at least one non-secure mode, the memory access request issued at said step (i) specifies a virtual address, said step (ii) is controlled by the non-secure operating system and one of said predetermined access control functions performed at said step (ii) comprises conversion of the virtual address to a physical address, the partition checking logic preventing at said step (iv) the access specified by that memory access request if the physical address generated by the memory management unit is within the secure memory.

22. A method as claimed in Claim 20 or Claim 21, wherein when the processor is operating in one of the at least one secure modes, the memory access request issued at said step (i) specifies a virtual address, said step (ii) is controlled by the secure operating system and one of said predetermined access control functions performed at said step (ii) comprises conversion of the virtual address to a physical address, the partition checking logic not being used in the at least one secure mode.

23. A method as claimed in Claim 22, wherein for all modes of operation of the processor, the memory access request issued at said step (i) specifies a virtual address, the partition checking logic being provided within a memory management unit used to perform said step (ii), and being operable whenever the processor is operating in said at least one non-secure mode.

27. A method as claimed in Claim 26, wherein the internal storage unit is a translation lookaside buffer (TLB) operable to store for a number of virtual address portions the corresponding physical address portions obtained from corresponding descriptors retrieved from the at least one table.

5

28. A method as claimed in claim 27, wherein the TLB is a micro-TLB, and the internal storage unit further comprises a main TLB for storing descriptors retrieved by the memory management unit from the at least one table, the method comprising the step of:

10 transferring access control information from the main TLB to the micro-TLB prior to use of that access control information at said step (ii) to perform the predetermined access control functions for the memory access request; and

when the processor is operating in said at least one non-secure mode, employing the partition checking logic to prevent the transfer of any access control information  
15 from the main TLB to the micro-TLB that would allow access to said secure memory.

29. A method as claimed in any of claims 26 to 28, wherein the at least one table comprises a non-secure table for use when the processor is operating in said at least one non-secure mode and containing descriptors generated by the non-secure operating  
20 system, in the event that a descriptor within that non-secure table is associated with a memory region that at least partially incorporates a part of the secure memory, the method comprising the step of:

when the processor is operating in non-secure mode, employing the partition checking logic to prevent the internal storage unit from storing as access control  
25 information the physical address portion specified by that descriptor if the physical address that would then be produced for the virtual address is within the secure memory.

30. A method as claimed in Claim 29 when dependent on Claim 28, wherein the at least one table further comprises a secure table within the secure memory that contains  
30 descriptors generated by the secure operating system, the main TLB comprising a flag associated with each descriptor stored within the main TLB, and the method comprising the step of:



36. A method as claimed in Claim 35, wherein the device has a predetermined pin for outputting the domain signal onto the device bus.

37. A method as claimed in any of claims 20 to 36, wherein the processor is coupled  
5 to a system bus, and a part of the memory comprises a tightly coupled memory connected to the system bus, the method comprising the steps of:

defining in a control register the physical address range for the tightly coupled memory; and

10 setting, by the processor when operating in a privileged secure mode, a control flag to indicate whether the tightly coupled memory is controllable by the processor only when executing in a privileged secure mode or is controllable by the processor when executing in the at least one non-secure mode.

38. A method as claimed in Claim 37, wherein if the tightly coupled memory is  
15 controllable by the processor when executing in the at least one non-secure mode, secure data is prevented from being stored in the tightly coupled memory.

39. A data processing apparatus, substantially as hereinbefore described with  
reference to Figures 21 to 32.

20

40. A method of controlling access to a memory in a data processing apparatus, substantially as hereinbefore described with reference to Figures 21 to 32.

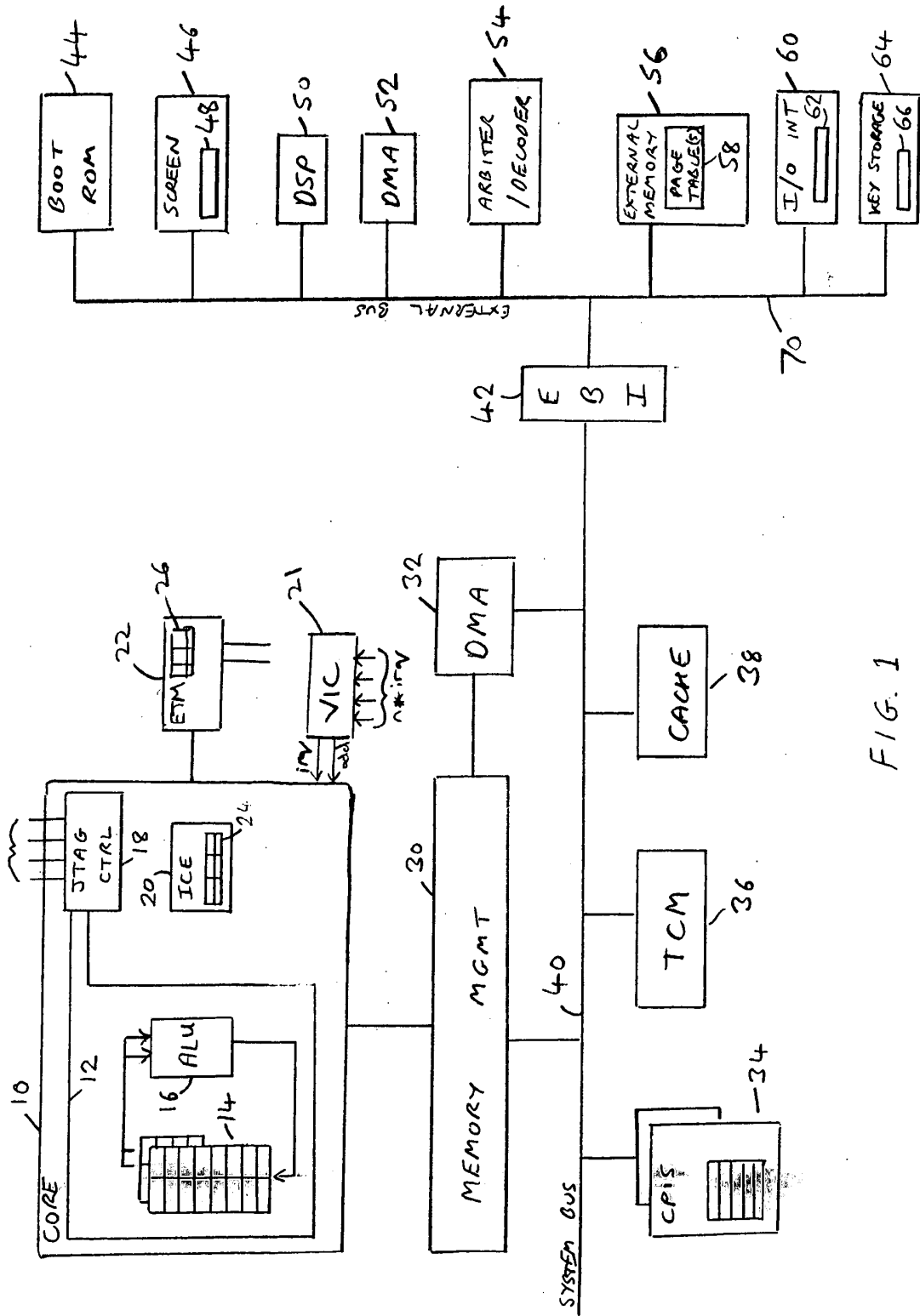


FIG. 1

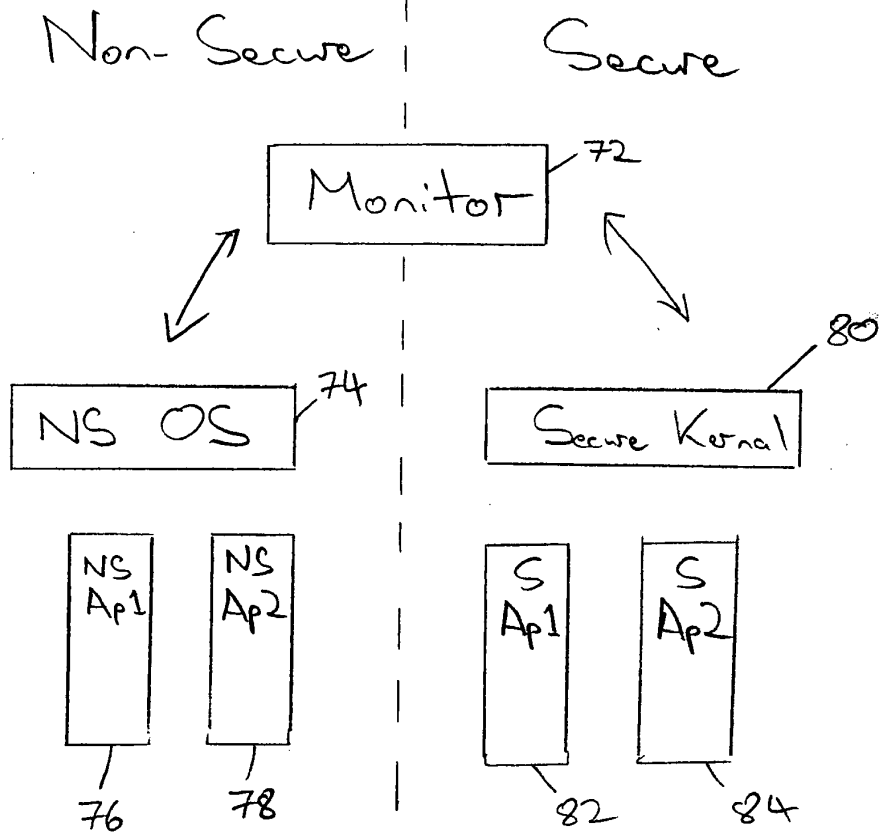


Fig. 2

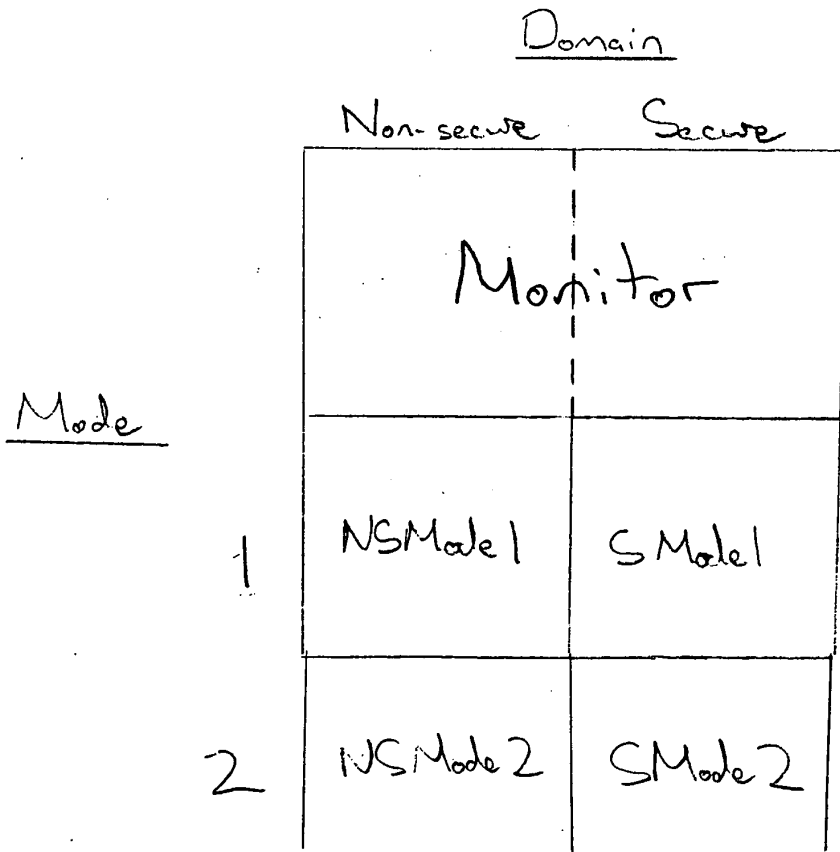


Fig. 3

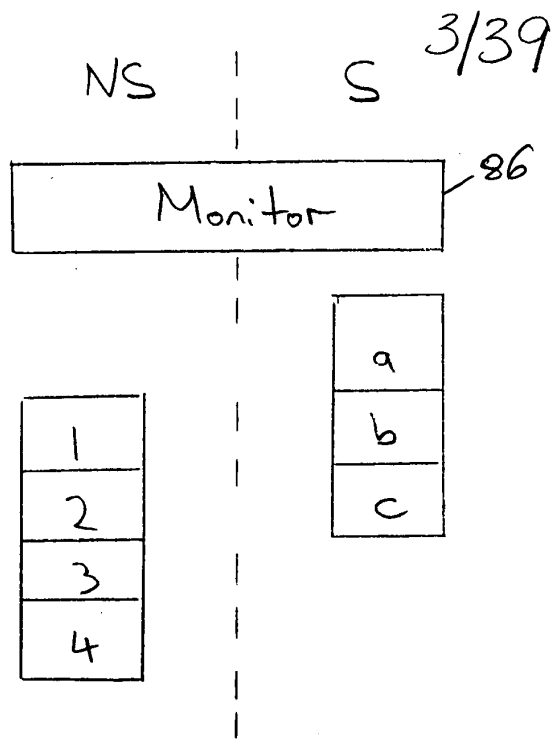


Fig. 4

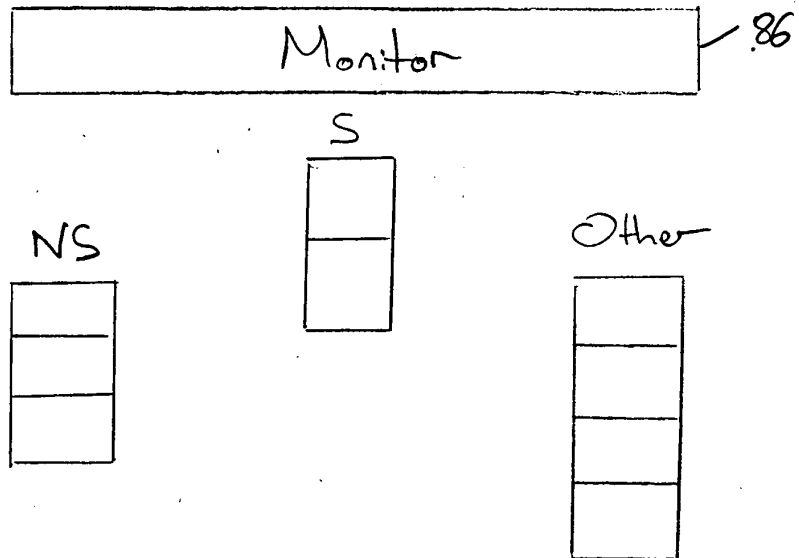
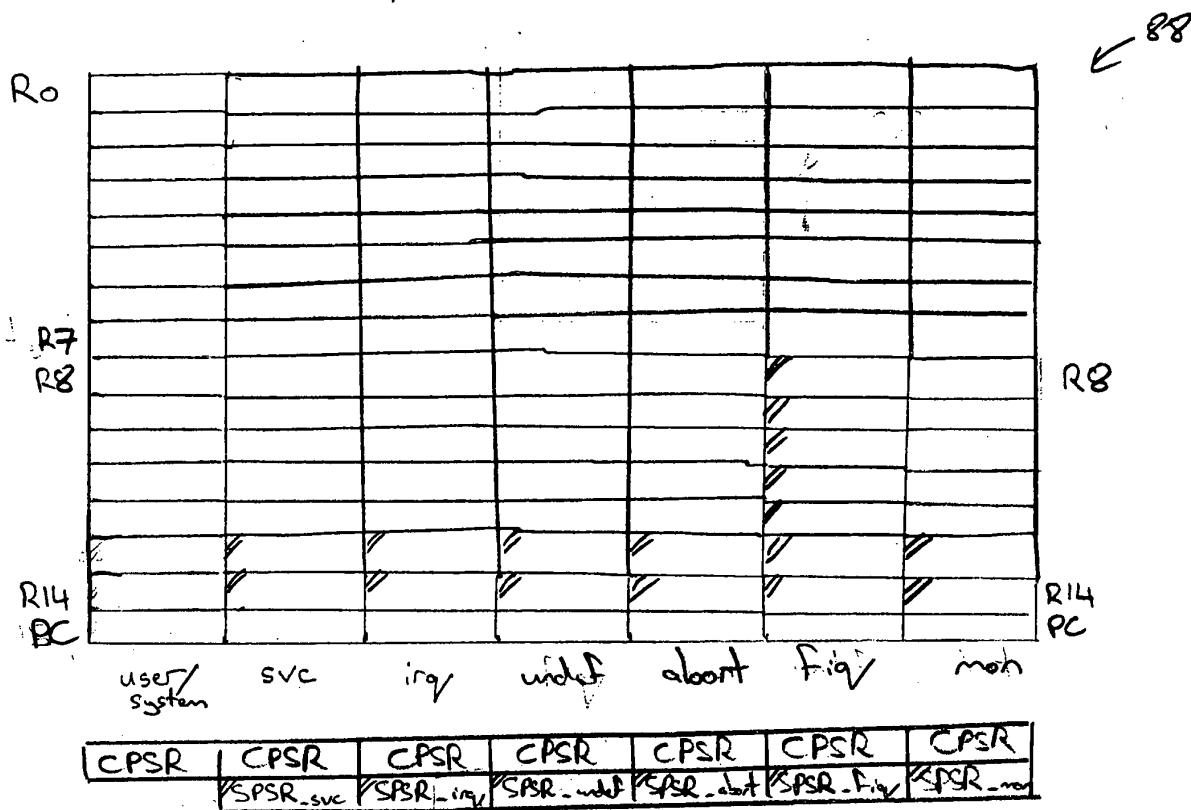


Fig. 5

4/39



// = private to mode      Fig. 6

88 ↘

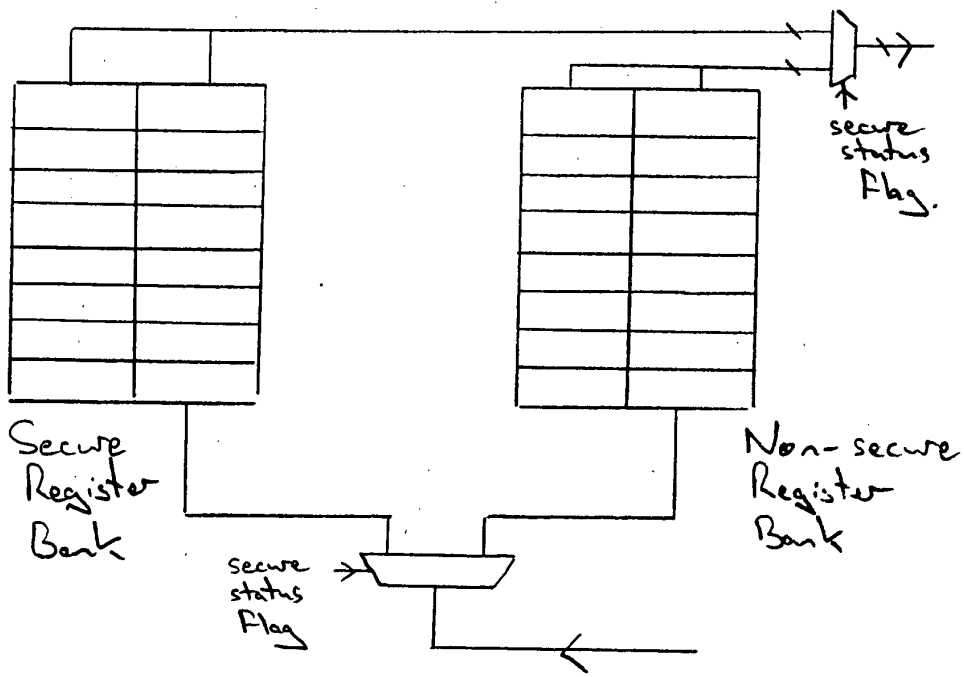


Fig. 7

5/39

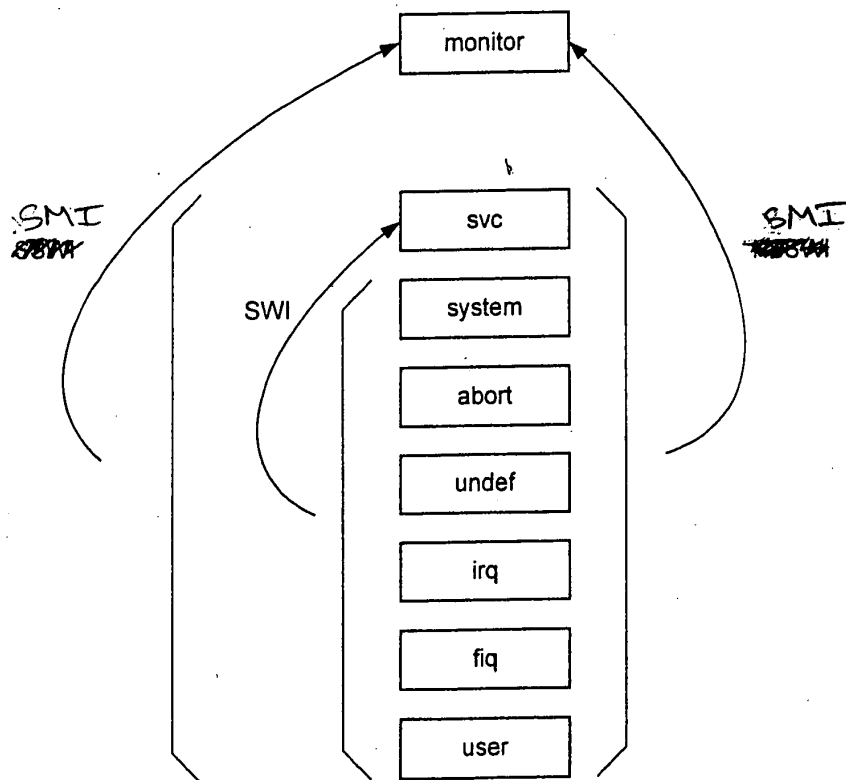


Fig. 8

6/39

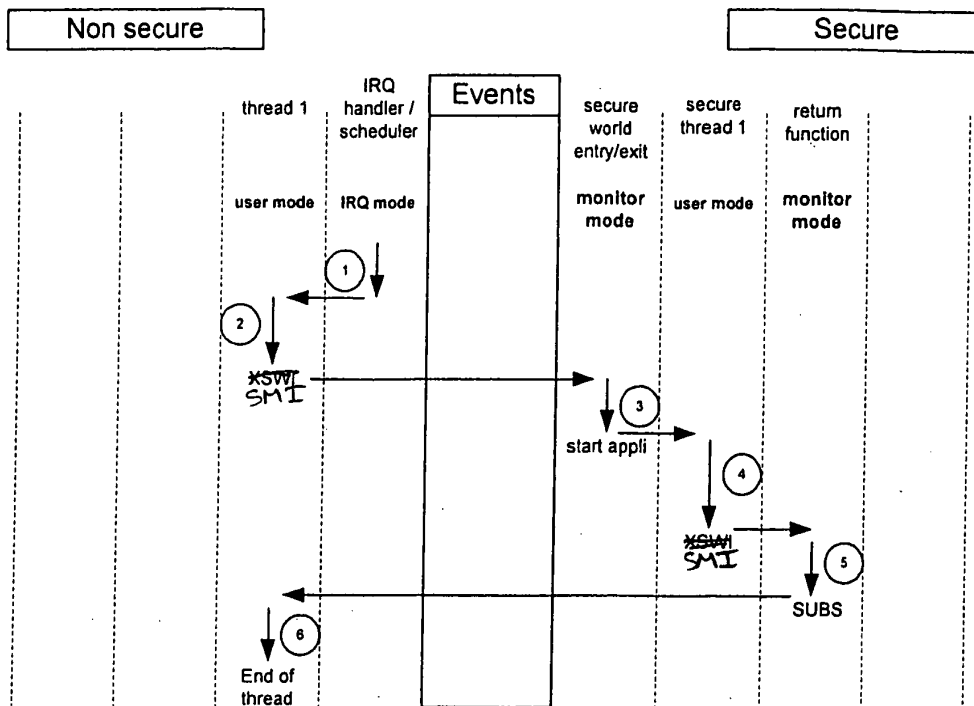


Fig. 9

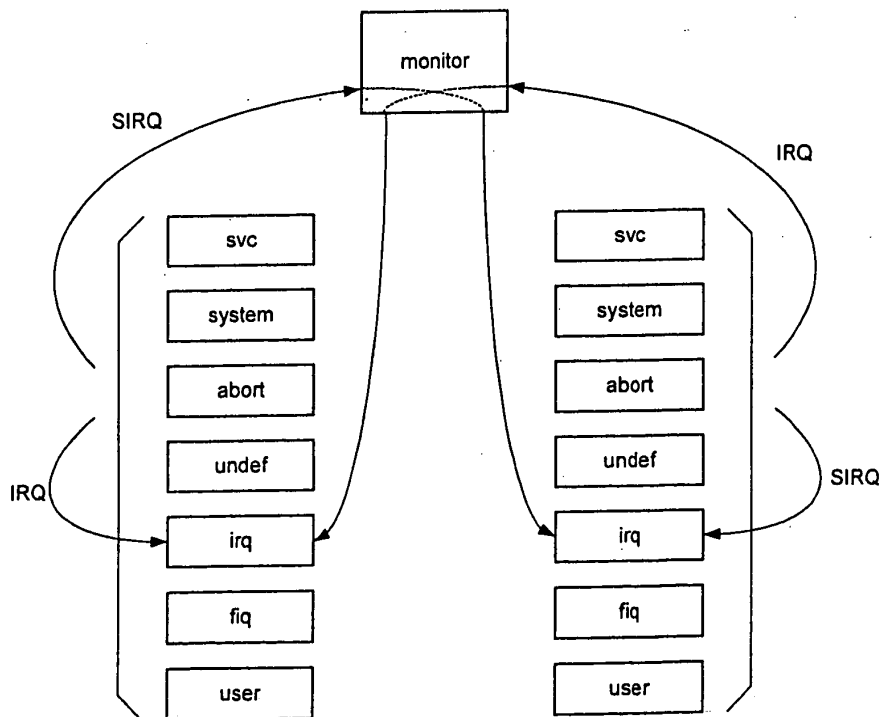


Fig. 10

[illegible]



8/39

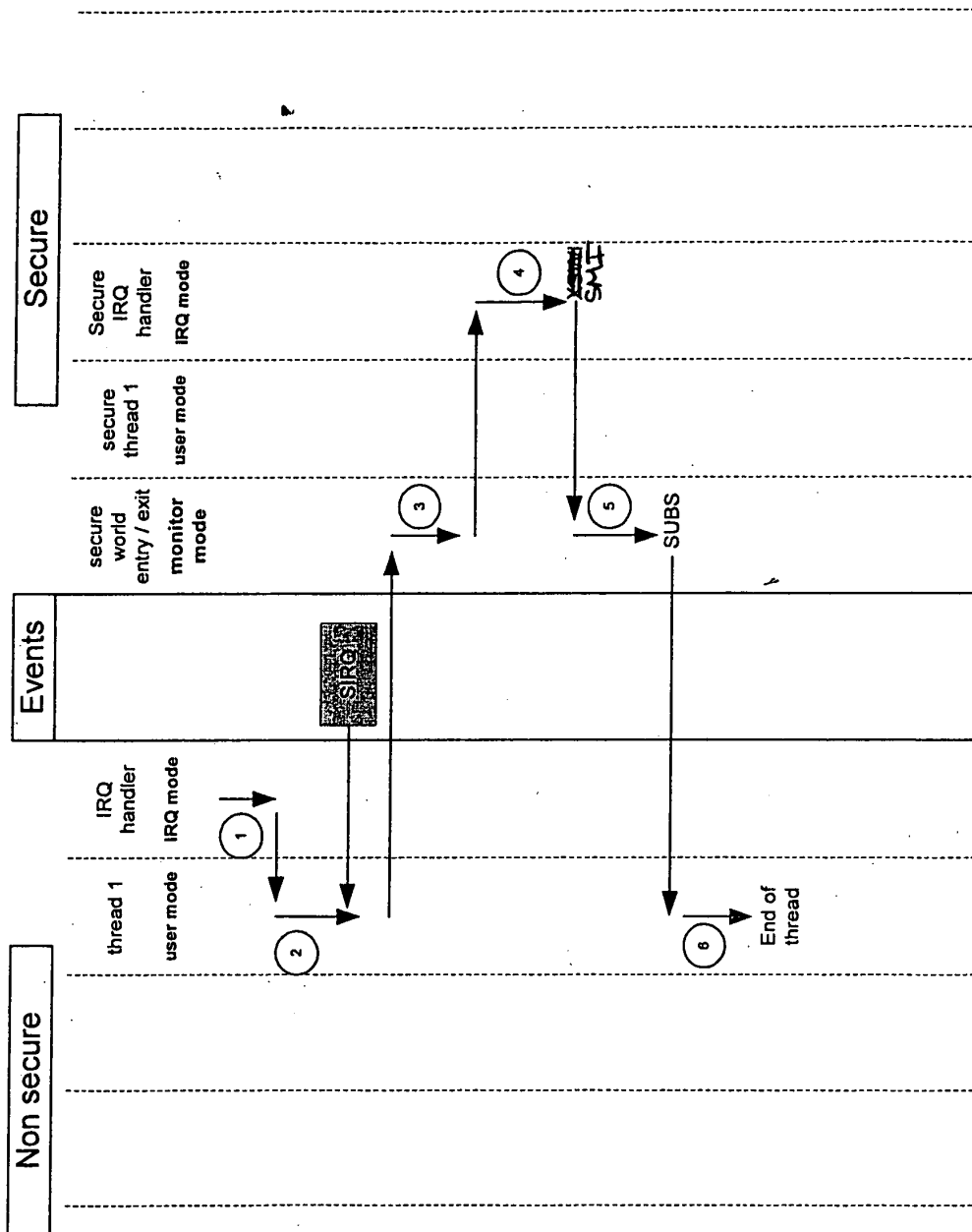


Fig. 11B

The diagram illustrates the monitor abstraction for the ARMv7-M architecture. It features a central 'monitor' block at the top. Below it are two identical vertical stacks of state blocks, each enclosed in a large bracket. The left stack contains blocks for 'svc', 'system', 'abort', 'undef', 'irq', 'fiq', and 'user' from top to bottom. The right stack contains blocks for 'svc', 'system', 'abort', 'undef', 'irq', 'fiq', and 'user' from top to bottom. Arrows indicate the flow of control: from the 'irq' block in the left stack to the 'monitor', and from the 'monitor' to the 'irq' block in the right stack. Similarly, from the 'irq' block in the right stack to the 'monitor', and from the 'monitor' to the 'irq' block in the left stack. Additionally, there are self-loop arrows on the 'irq' blocks in both stacks, labeled 'IRQ' on the left and 'SIRQ' on the right. A curved arrow labeled 'SIRQ' also points from the 'irq' block in the right stack to the 'irq' block in the left stack.

The diagram illustrates the execution flow of thread 1 across three domains: Non secure, Events, and Secure.

- Non secure domain:**
  - thread 1 user mode:** Starts at step 2, then continues after step 7.
- Events domain:**
  - IRQ handler IRQ mode:** Starts at step 1, then continues after step 3.
- Secure domain:**
  - secure thread 1 user mode:** Starts at step 4, then continues after step 6.
  - Secure IRQ handler IRQ mode:** Starts at step 5, then continues after step 7.

**Sequence of Events:**

- Step 1:** IRQ handler (IRQ mode) begins execution.
- Step 2:** Thread 1 (user mode) begins execution.
- Step 3:** Thread 1 (user mode) triggers a **SIRQ** (Secure IRQ) event.
- Step 4:** Thread 1 (user mode) triggers an **SMT** (Secure Monitor Trap) event.
- Step 5:** Secure thread 1 (user mode) begins execution.
- Step 6:** Secure thread 1 (user mode) triggers a **SMT** (Secure Monitor Trap) event.
- Step 7:** Thread 1 (user mode) triggers a **SUBS** (Subsidiary) event.
- Step 8:** Thread 1 (user mode) ends.

Fig. 13A

10/39

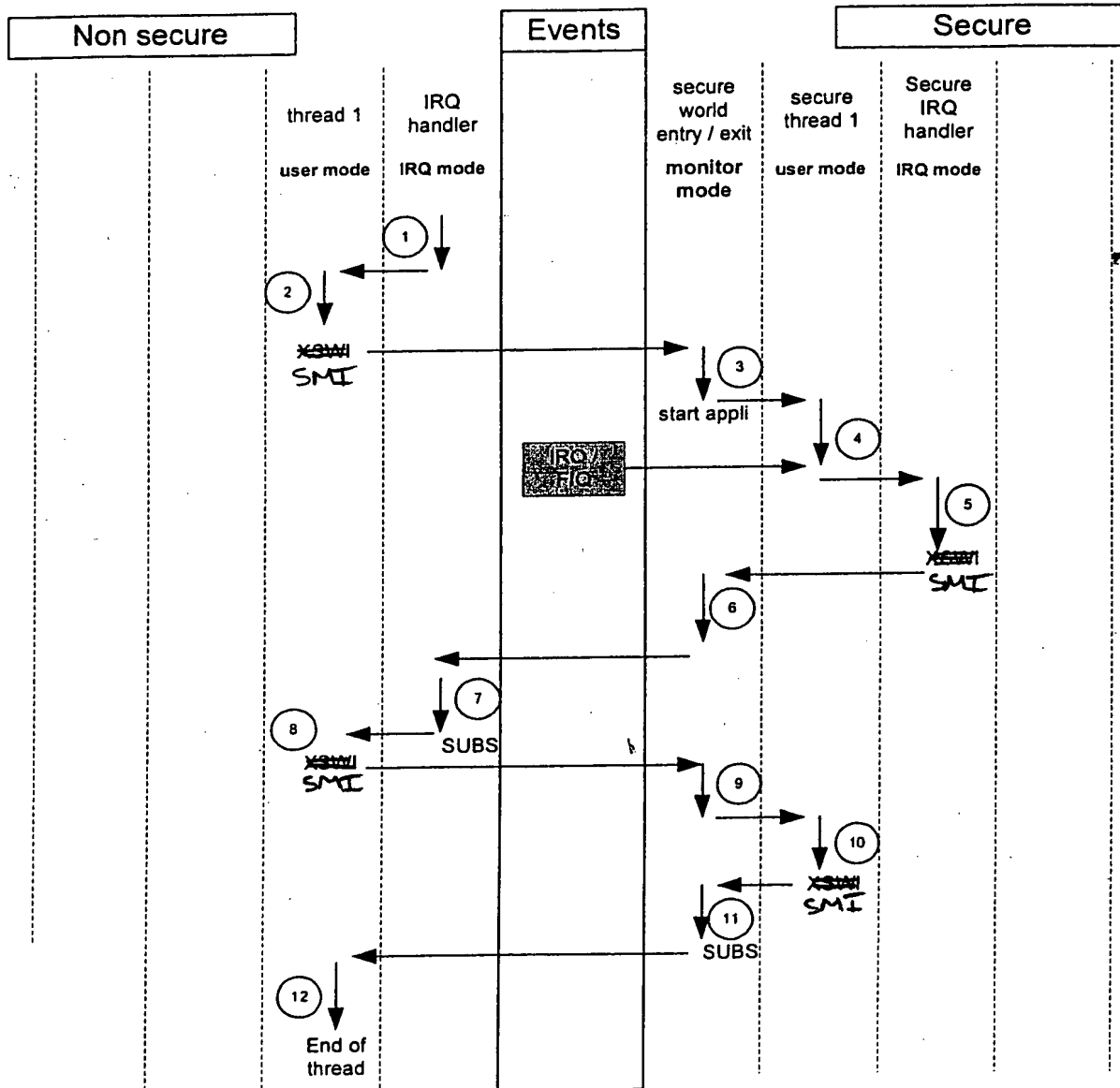


Fig. 13B

11/39

Exception	Vector offset	Corresponding mode
Reset	0x00	Supervisor mode
<del>Under</del>	0x04	Monitor mode / <del>Under mode</del>
SWI	0x08	Supervisor mode / Monitor mode
Prefetch abort	0x0C	Abort mode / Monitor mode
Data abort	0x10	Abort mode / Monitor mode
IRQ / SIRQ	0x18	IRQ mode / Monitor mode
FIQ	0x1C	FIQ mode / Monitor mode
SMT	0x14	<del>Under mode</del> / Monitor mode

Fig. 14

Secure

Monitor

Reset	VM0
<del>Under</del>	VM1
SWI	VM2
Prefetch abort	VM3
Data abort	VM4
IRQ / SIRQ	VM5
FIQ	VM6
SMT	VM7

Non-Secure

Reset	VS0
<del>Under</del>	VS1
SWI	VS2
Prefetch abort	VS3
Data abort	VS4
IRQ / SIRQ	VS5
FIQ	VS6
SMT	VS7

Reset	VNS0
<del>Under</del>	VNS1
SWI	VNS2
Prefetch abort	VNS3
Data abort	VNS4
IRQ / SIRQ	VNS5
FIQ	VNS6
SMT	VNS7

Fig. 15

CPI5 Exception Trap Mask Register (1) <sup>only NS</sup> world exceptions illustrated

0	1	1	1	1	0	1
Reset + SMT	SWI	Prefetch Abort	Data Abort	IRQ	SIRQ	FIQ

1 = Mon(S)

0 = NS



OR via hardware / external

pin

value

Fig. 16

12/39

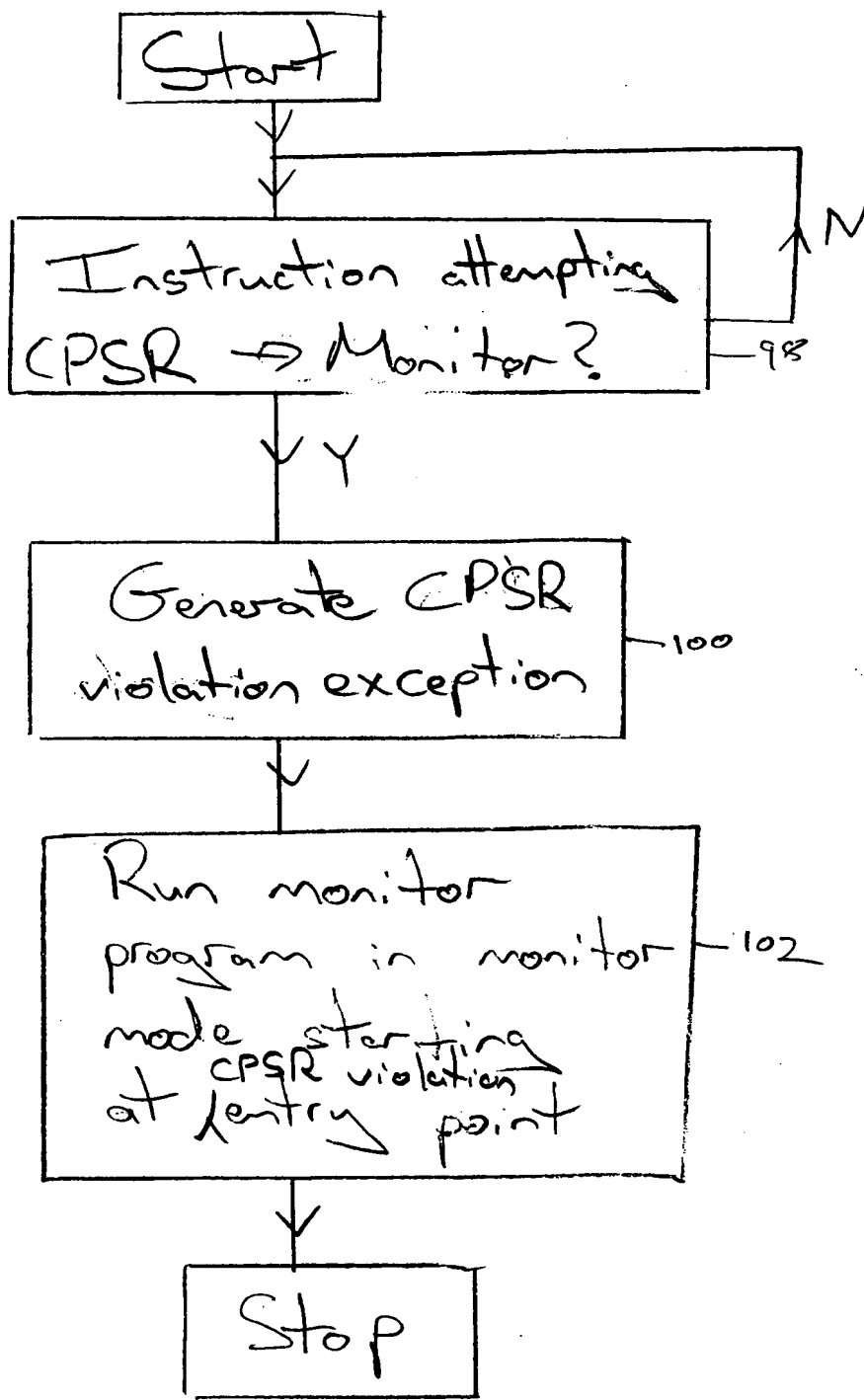


Fig. 17

13/39

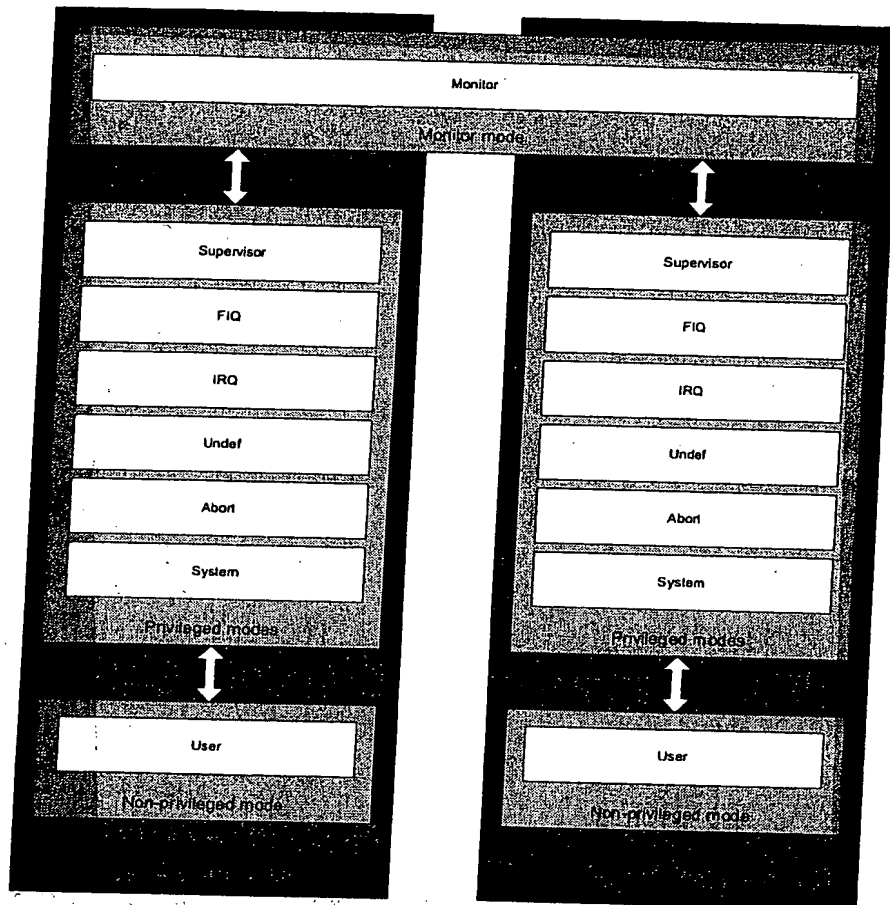


FIGURE 18

14/39

User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt	Monitor
R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq	R8
R9	R9	R9	R9	R9	R9	R9_fiq	R9
R10	R10	R10	R10	R10	R10	R10_fiq	R10
R11	R11	R11	R11	R11	R11	R11_fiq	R11
R12	R12	R12	R12	R12	R12	R12_fiq	R12
R13	R13	<del>R13_svc</del>	<del>R13_abt</del>	R13_und	R13_irq	R13_fiq	<del>R13_mon</del>
R14	R14	<del>R14_svc</del>	<del>R14_sbt</del>	R14_und	R14_irq	R14_fiq	<del>R14_mon</del>
PC	PC	PC	PC	PC	PC	PC	PC

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	SPSR_mon

FIGURE 19

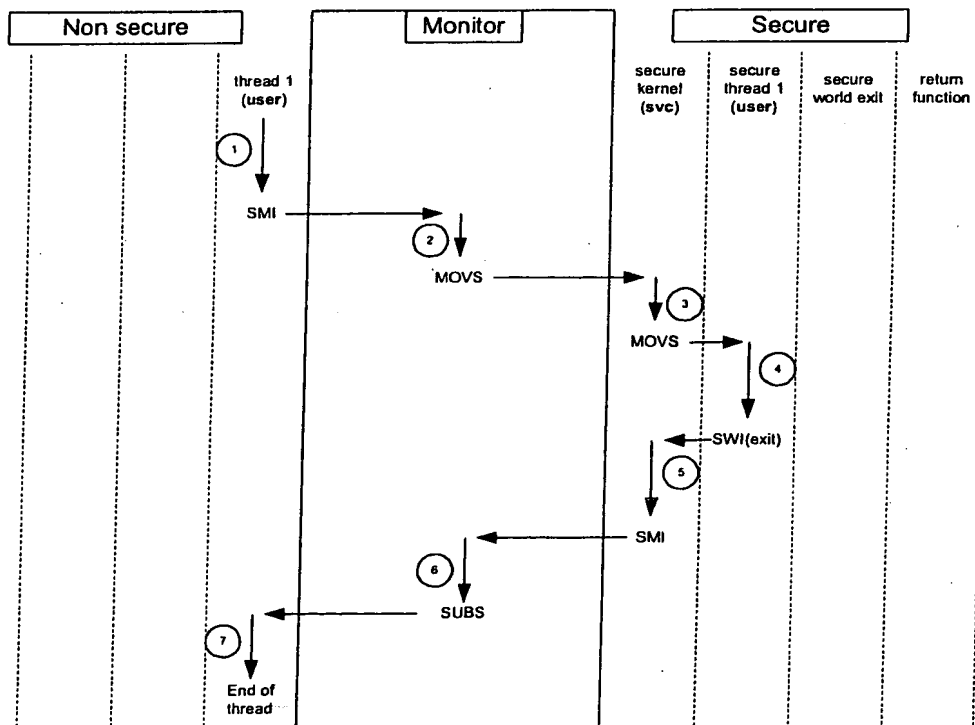


FIGURE 20

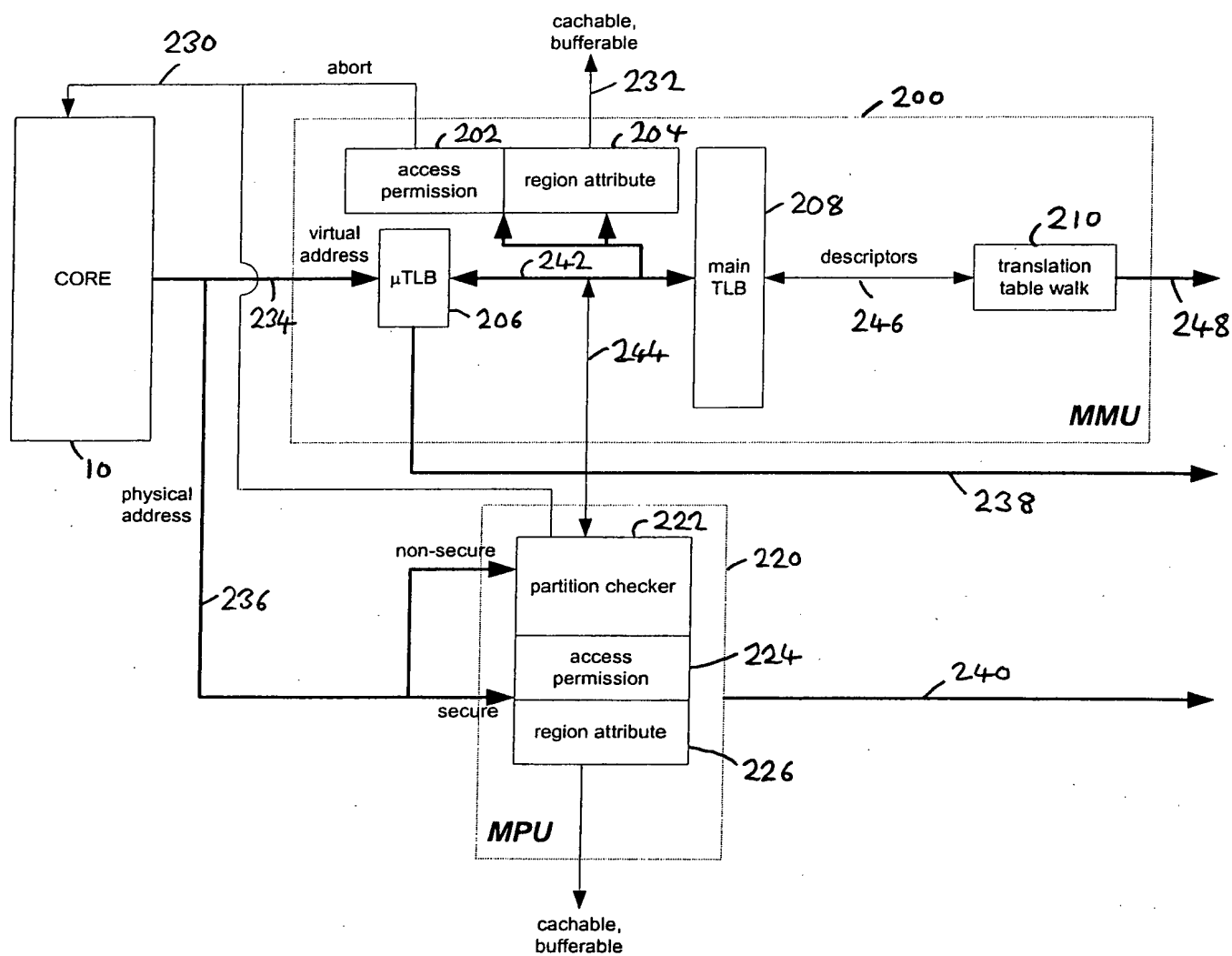


FIG. 21



16/39

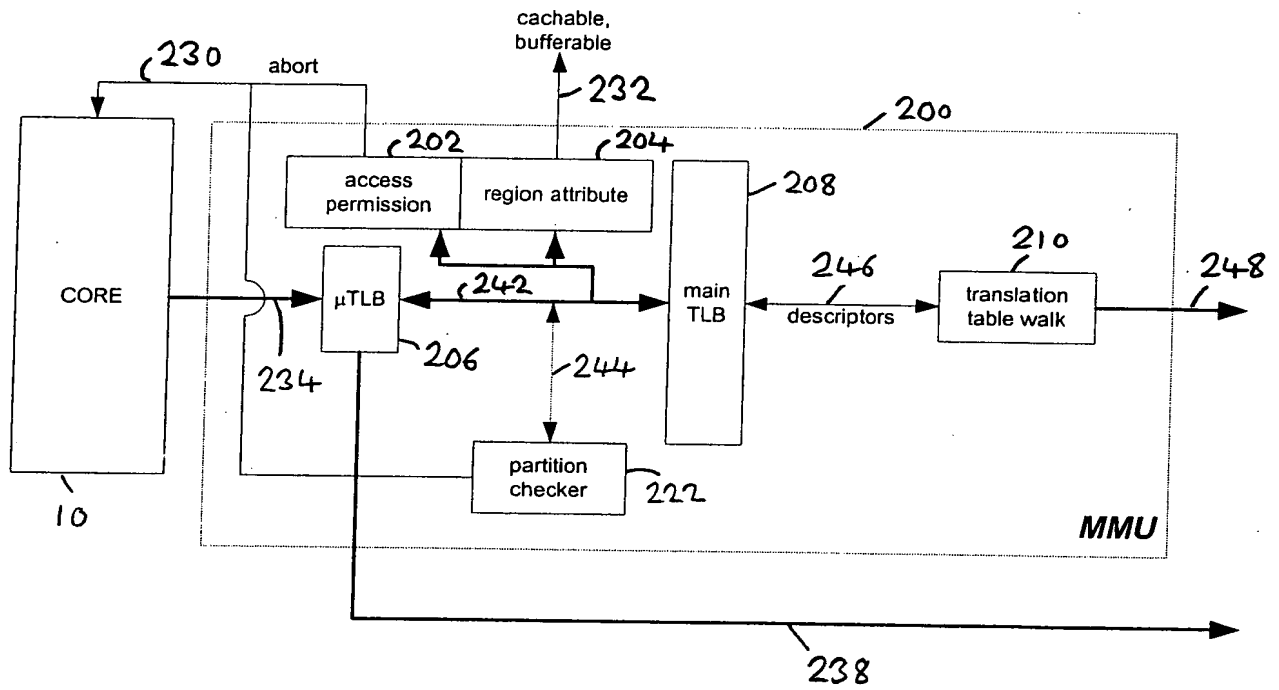


FIG. 22

17/39

program generates  
virtual address (VA)

300

secure

non-secure

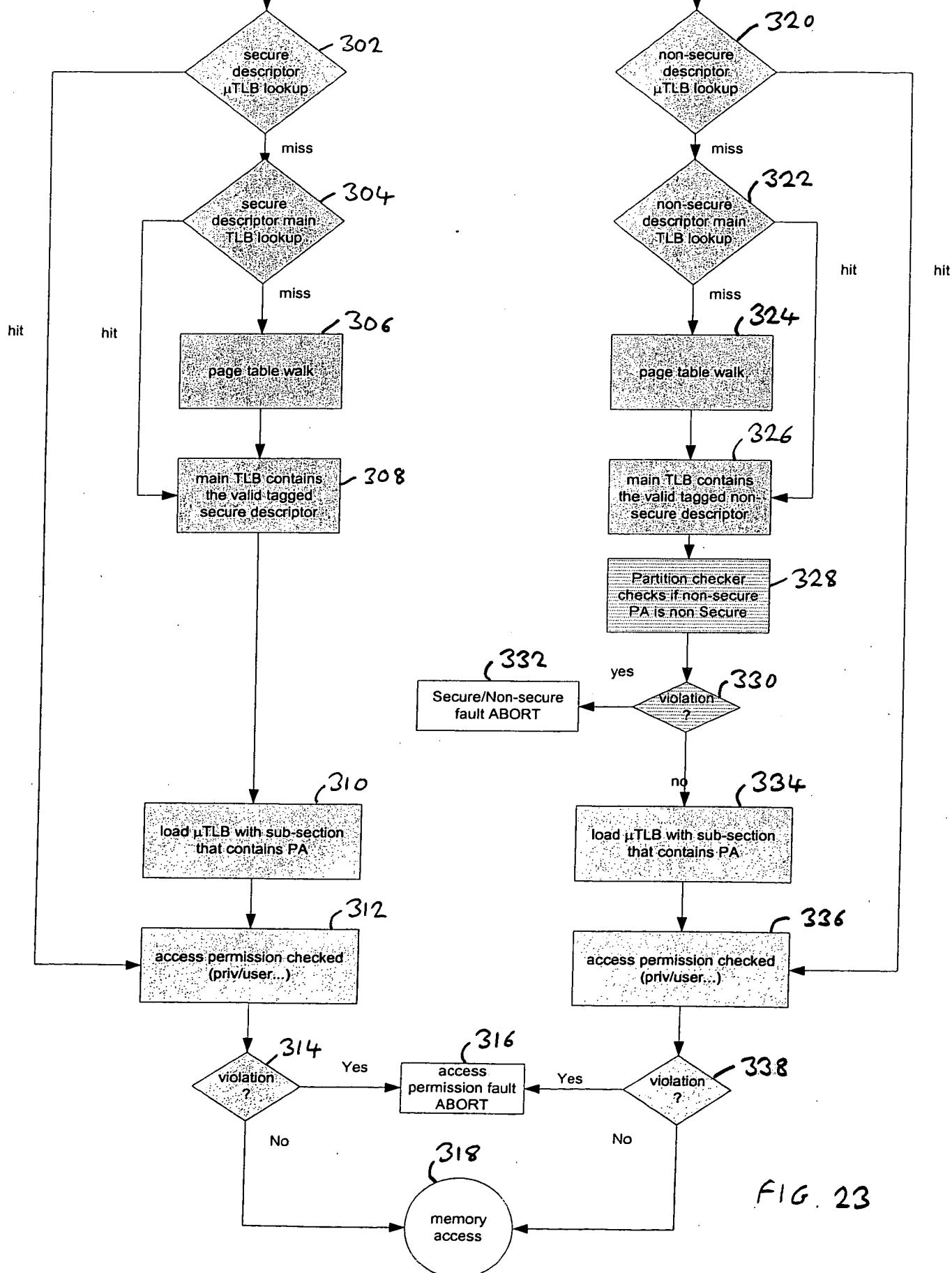


FIG. 23

18/39

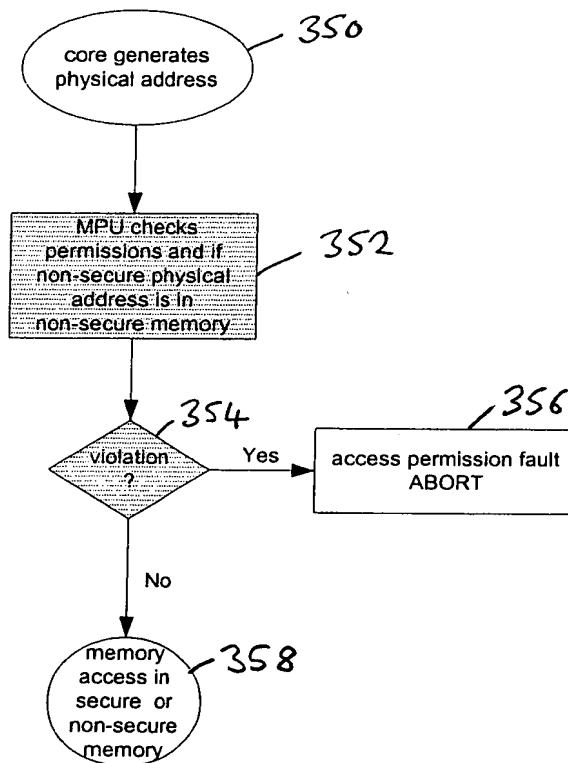


FIG. 24

19/39

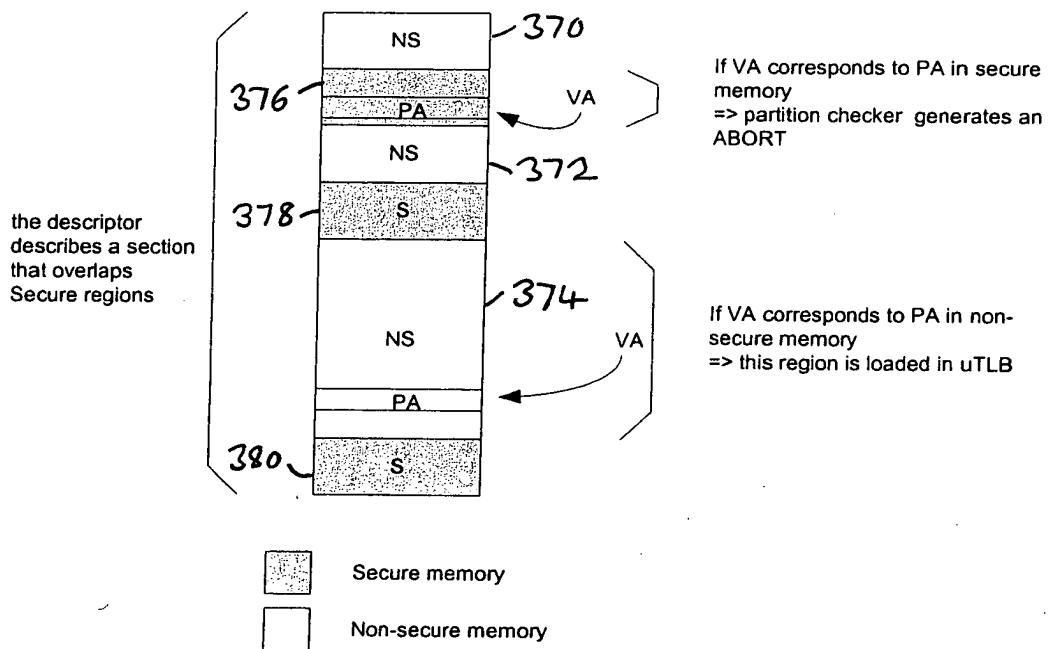


FIG. 25

20/39

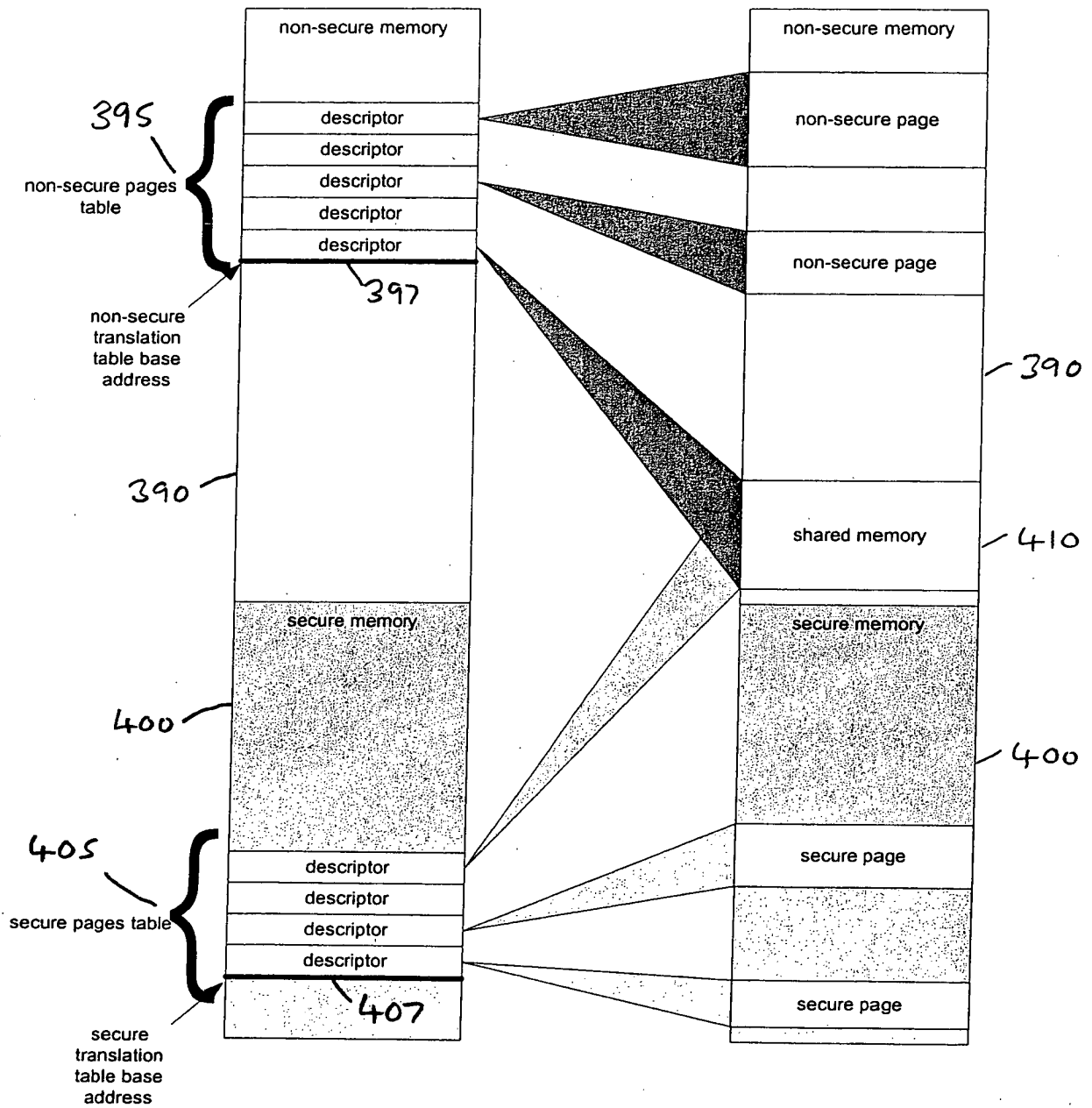


FIG. 26

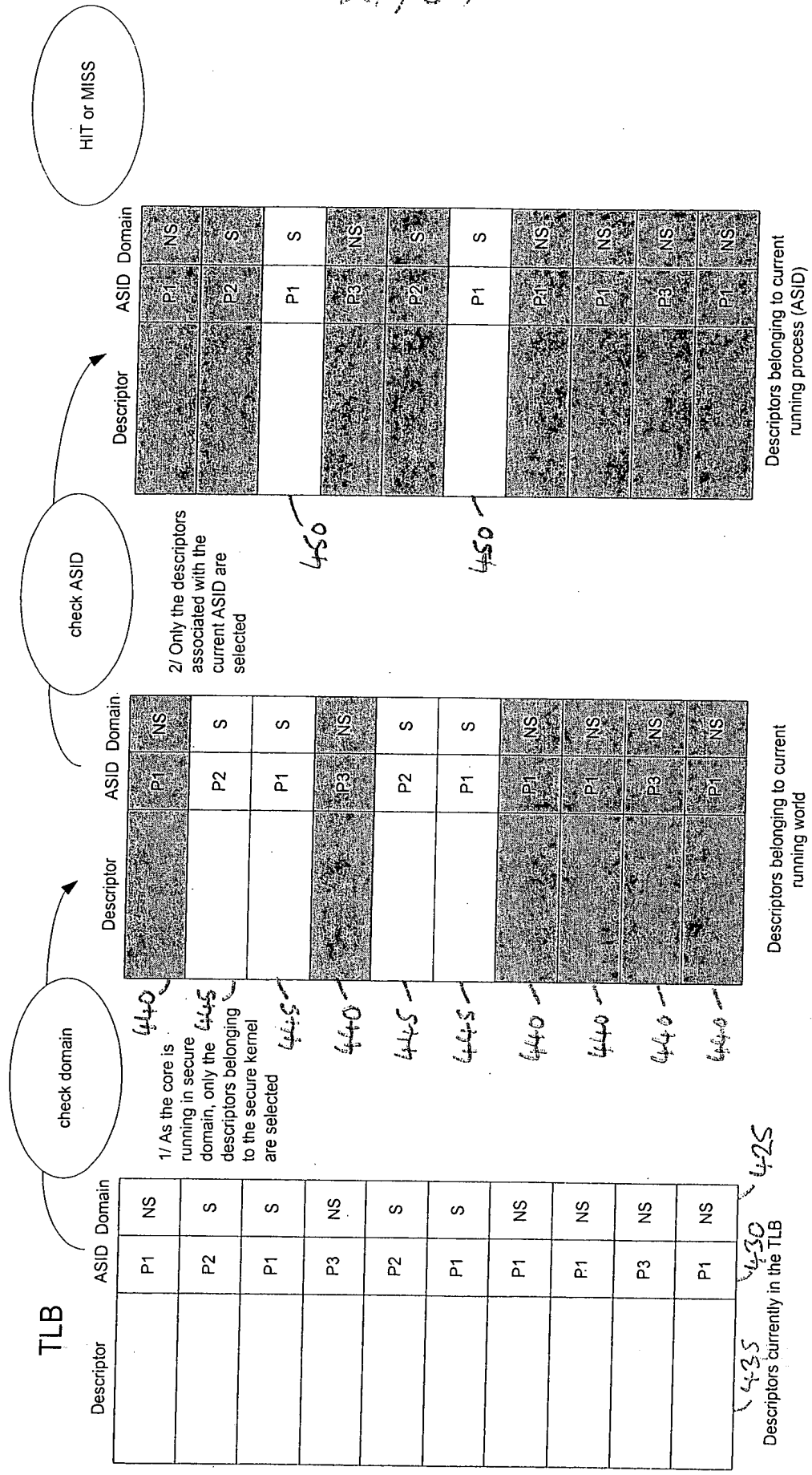


FIG. 27

208

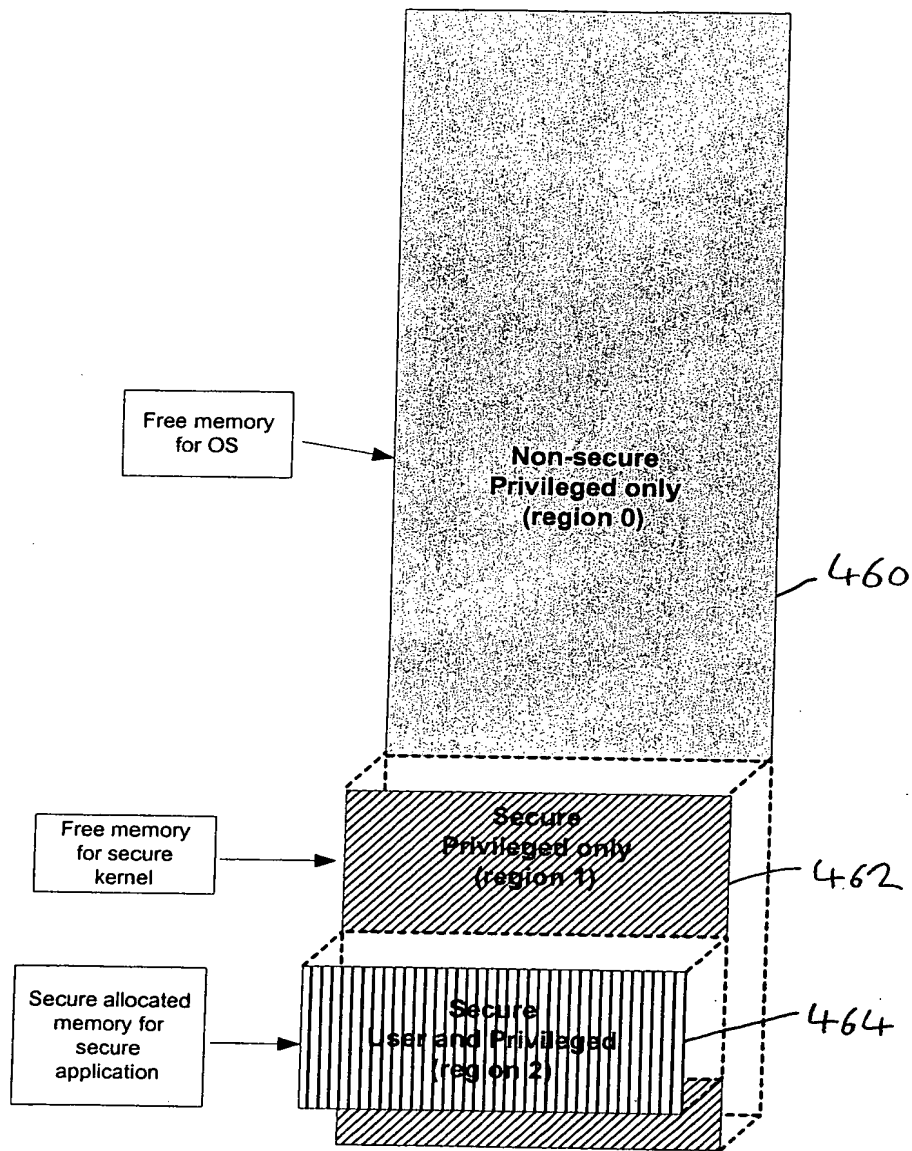


FIG. 28

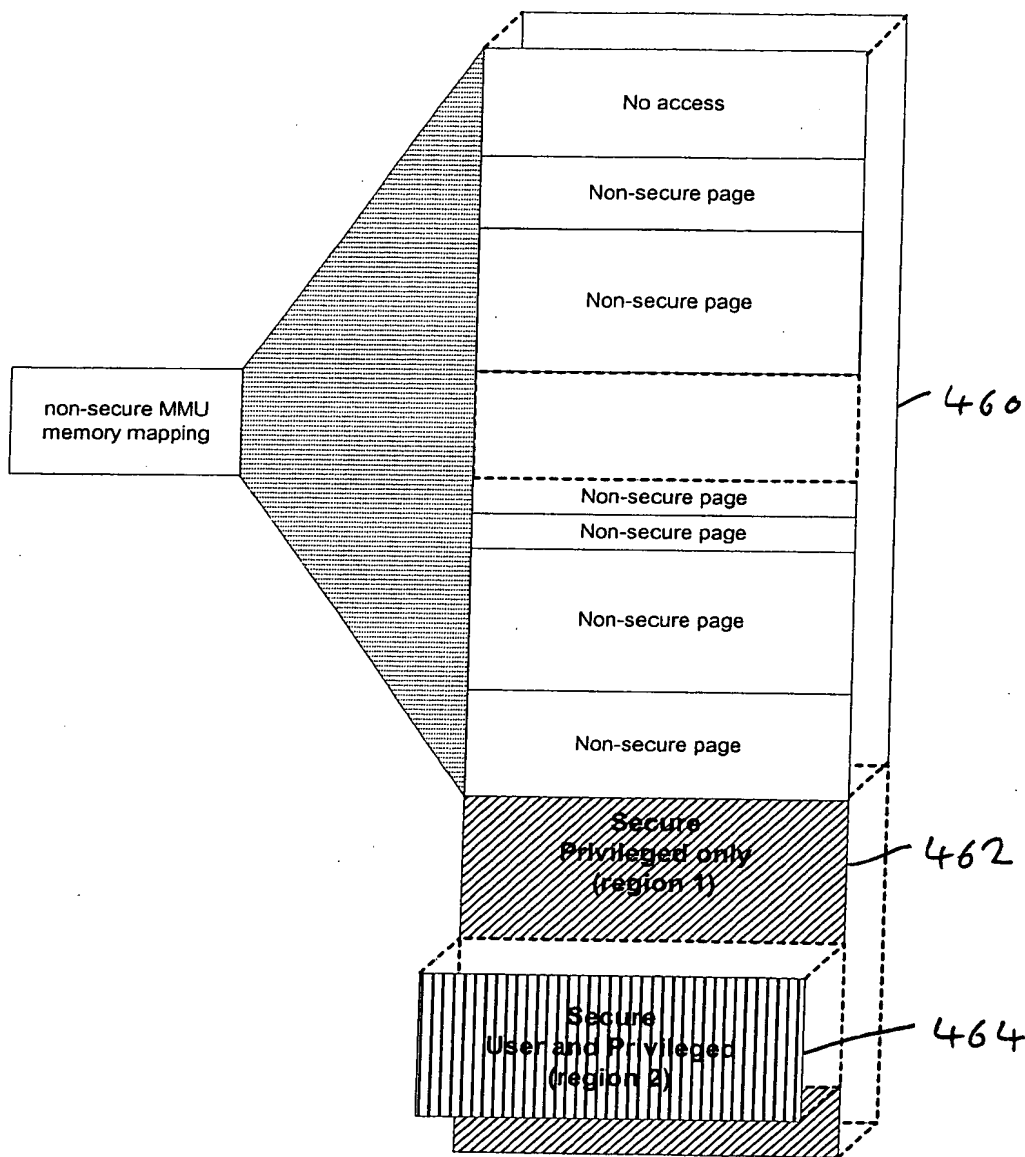


FIG. 29



24/39

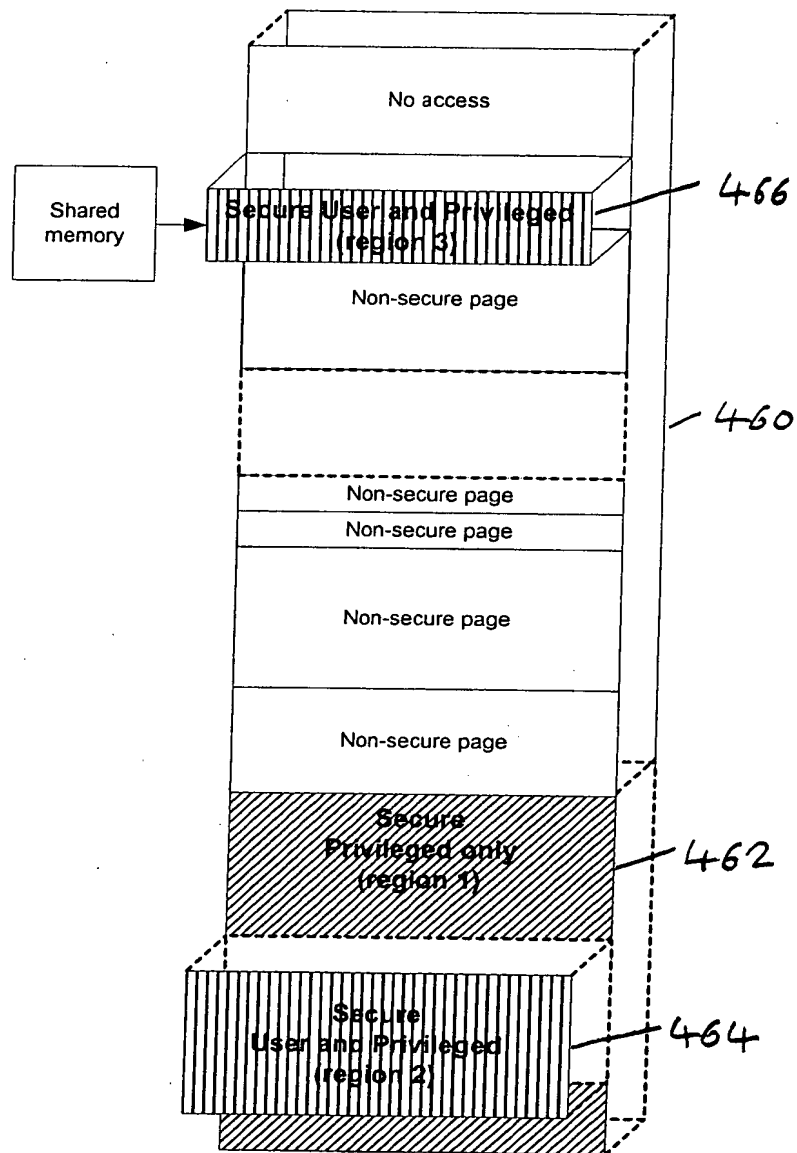
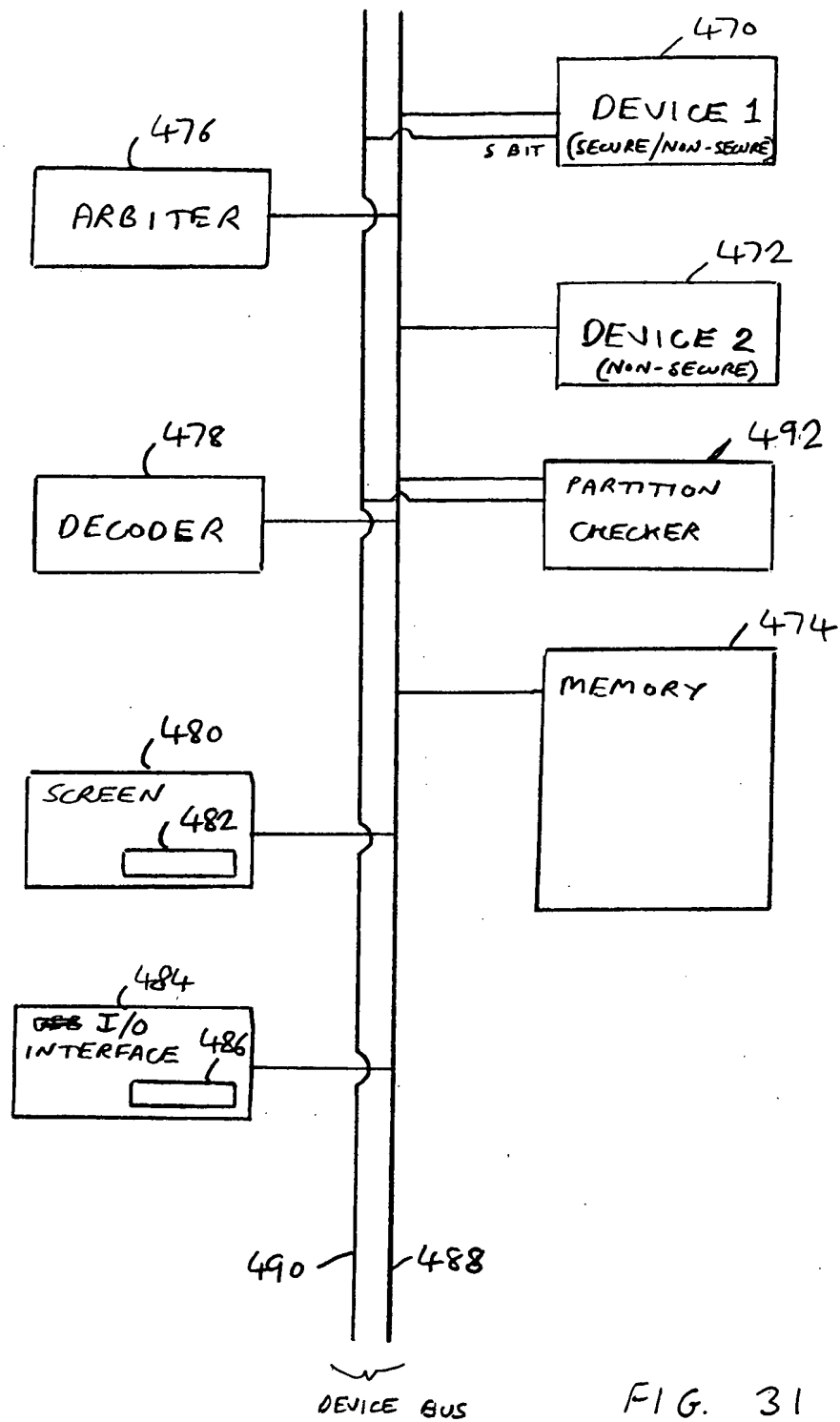


FIG. 30

25/39



26/39

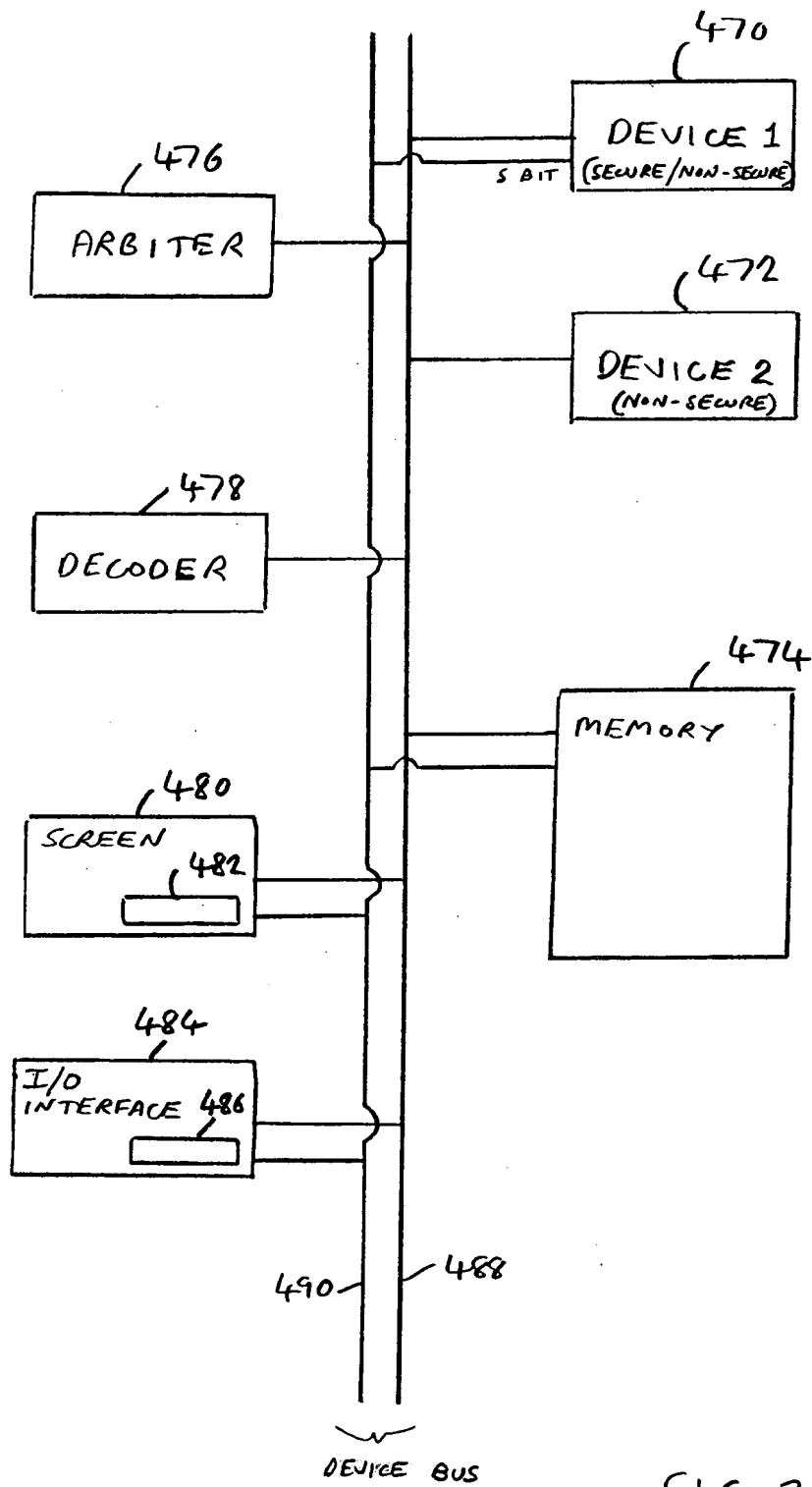


FIG. 32

27/39

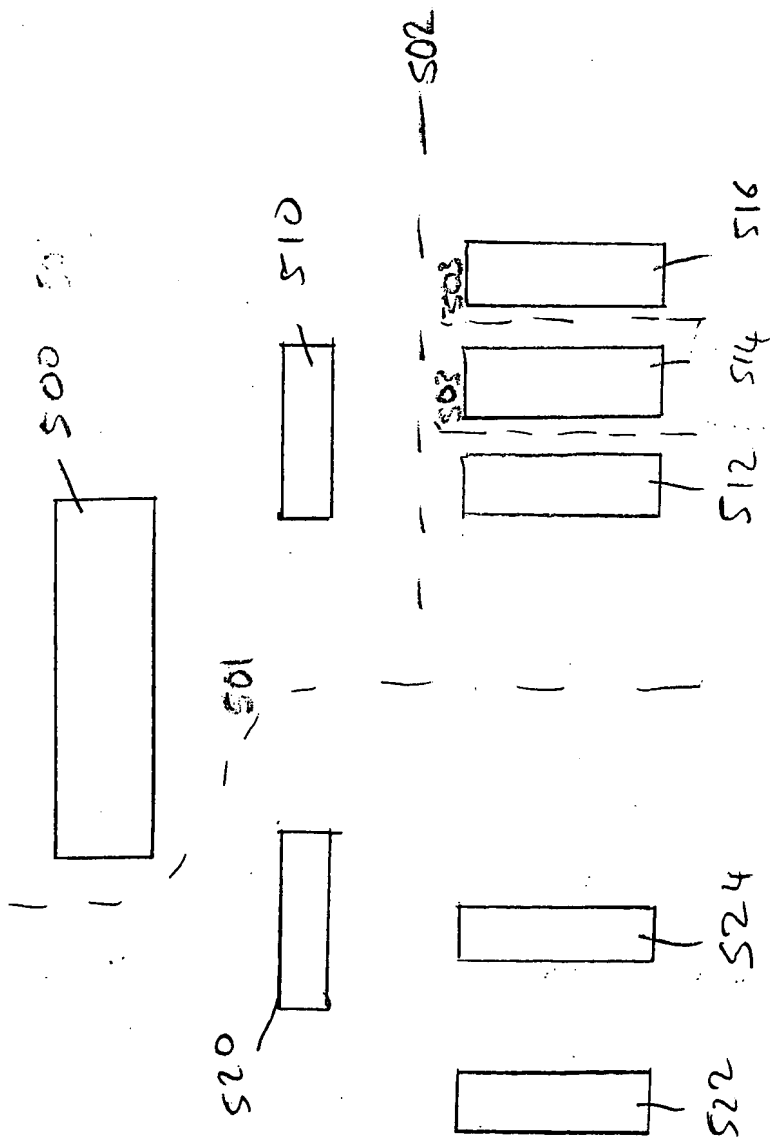


Figure 33

28/39

Method of entry	How to program?	How to enter?	Entry mode
Breakpoint hits	Debug TAP or software (CP14)	Program breakpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID <sup>(2)</sup> .	Halt/monitor <sup>(1)</sup>
Software breakpoint instruction	Put a BKPT instruction into scan chain 4 (Instruction Transfer Register) through Debug TAP or Use BKPT instruction directly in the code.	BKPT instruction must reach execution stage.	Halt/monitor
Vector trap breakpoint	Debug TAP	Program vector trap register and address matches.	Halt/monitor
Watchpoint hits	Debug TAP or software (CP14)	Program watchpoint register and/or context-ID register and comparisons succeed with Instruction Address and/or CP15 Context ID <sup>(2)</sup> .	Halt/monitor <sup>(1)</sup>
Internal debug request	Debug TAP	Halt instruction has been scanned in.	Halt
External debug request	Not applicable	EDBGRQ input pin is asserted.	Halt

<sup>(1)</sup>: In monitor mode, breakpoints and watchpoints cannot be data-dependent.

<sup>(2)</sup>: The cores have support for thread-aware breakpoints and watchpoints, in order to enable secure debug on some particular threads.




Figure 34

29/39

Name	Meaning	Reset value	Access	Inserted in scan chain for test
Monitor mode enable bit	0: halt mode 1: monitor mode	1	R/W by programming the ICE by the JTAG (scan1) ▪ R/W by using MRC/MCR instruction (CP14)	yes
Secure debug enable bit	0: debug in non-secure world only. 1: debug in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1)	no
Secure trace enable bit	0: ETM is enabled in non-secure world only. 1: ETM is enabled in secure world and non-secure world	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1)	no
Secure user-mode enable bit	0: debug is not possible in secure user mode 1: debug is possible in secure user mode	1	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1)	no
Secure thread-aware enable bit	0: debug is not possible for a particular thread 1: debug is possible for a particular thread	0	In functional mode or debug monitor mode: R/W by using MRC/MCR instruction (CP14) (only in secure supervisor mode)  In Debug halt mode: No access – MCR/MRC instructions have any effect.  (R/W by programming the ICE by the JTAG (scan1) if JSDAEN=1)	no

Figure 35

Function Table

D	CK	Q[n+1]
0		0
1		1
X		Q[n]

Logic Symbol

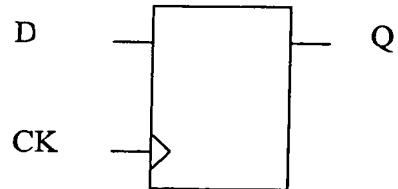





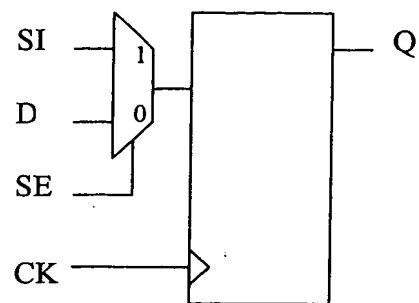


FIGURE 36

**Function Table**

D	SI	SE	CK	Q[n+1]
0	X	0		0
1	X	0		1
X	X	X		Q[n]
X	0	1		0
X	1	1		1

**Logic Symbol**



**Figure 37**



32/39

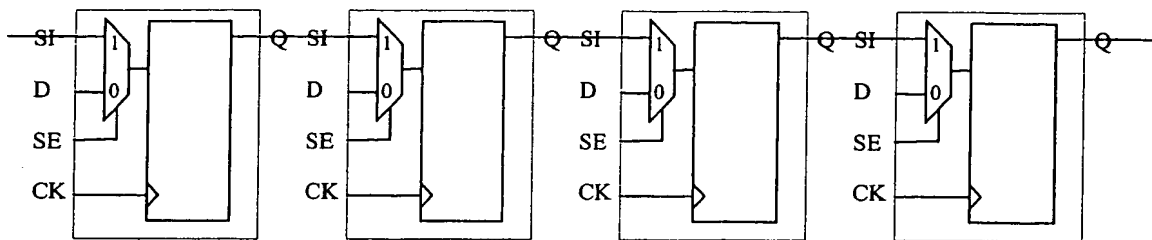


FIGURE 38

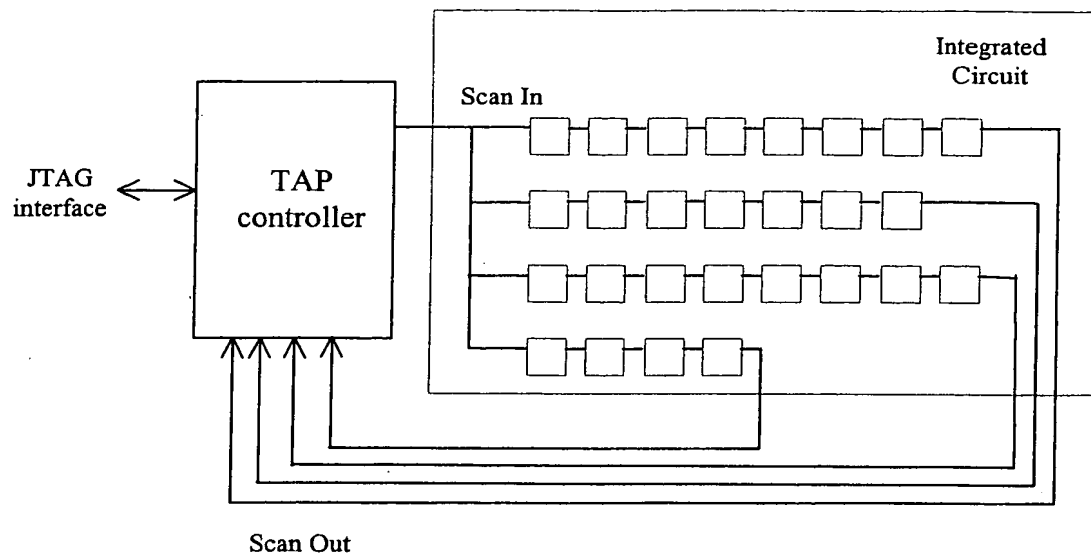


FIGURE 39

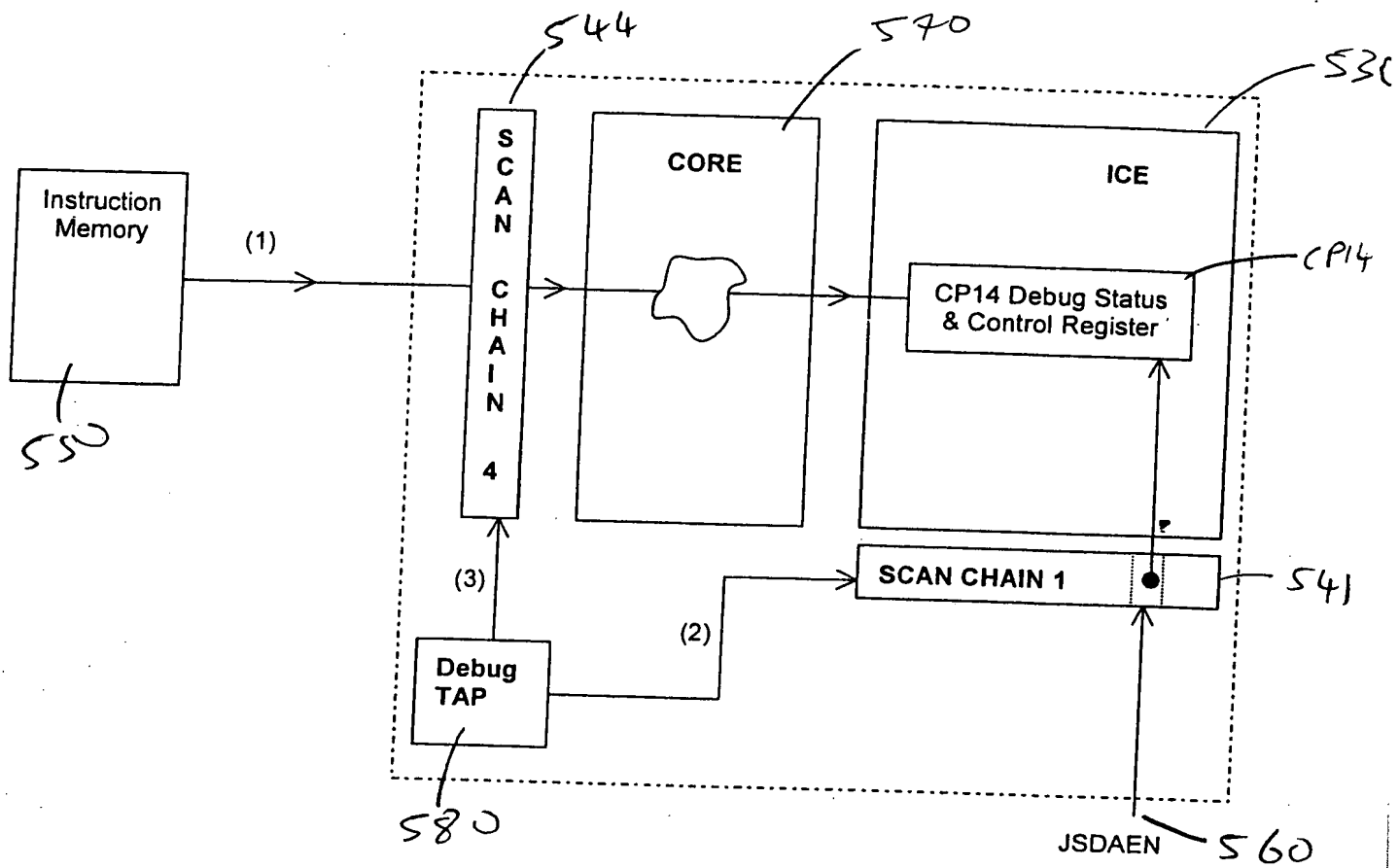


Figure 41

35/39

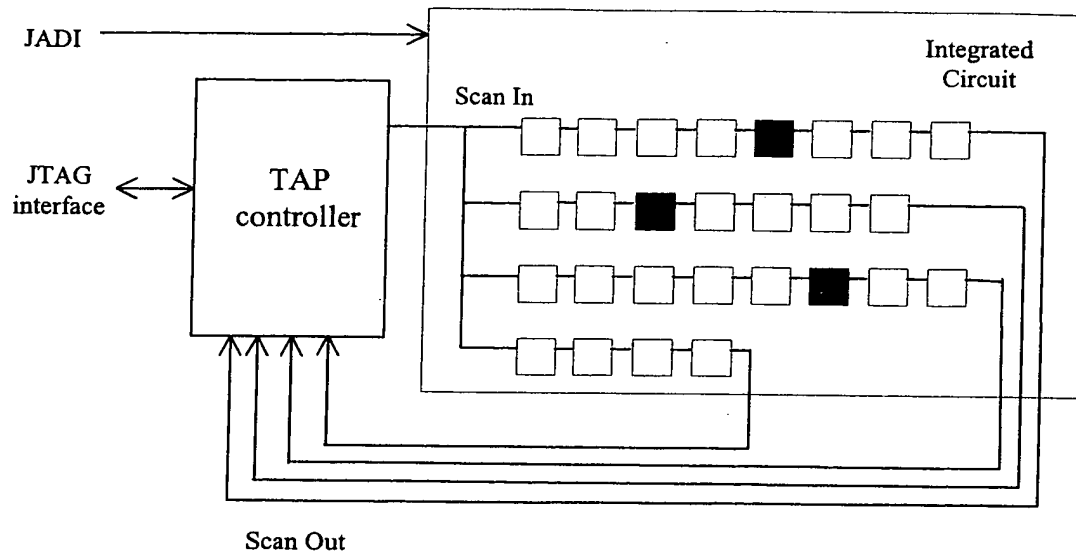


FIGURE 40A

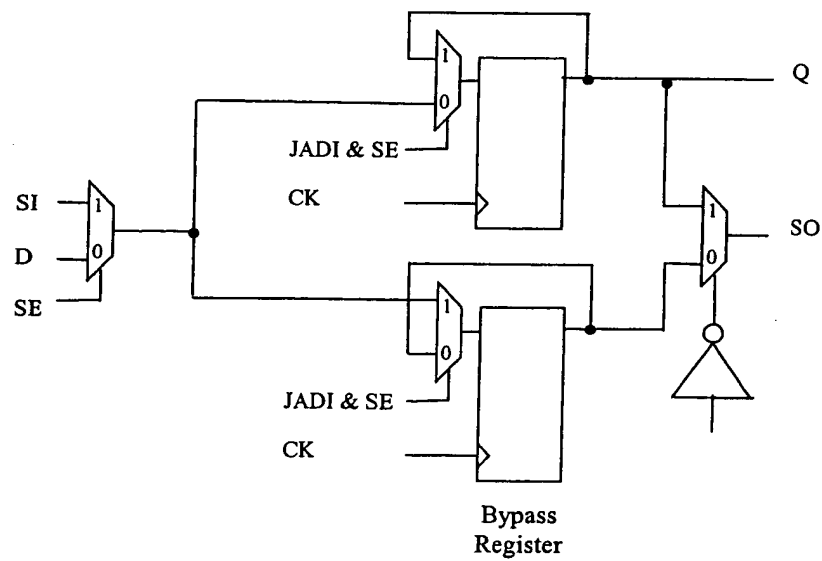


FIGURE 40B

36/39

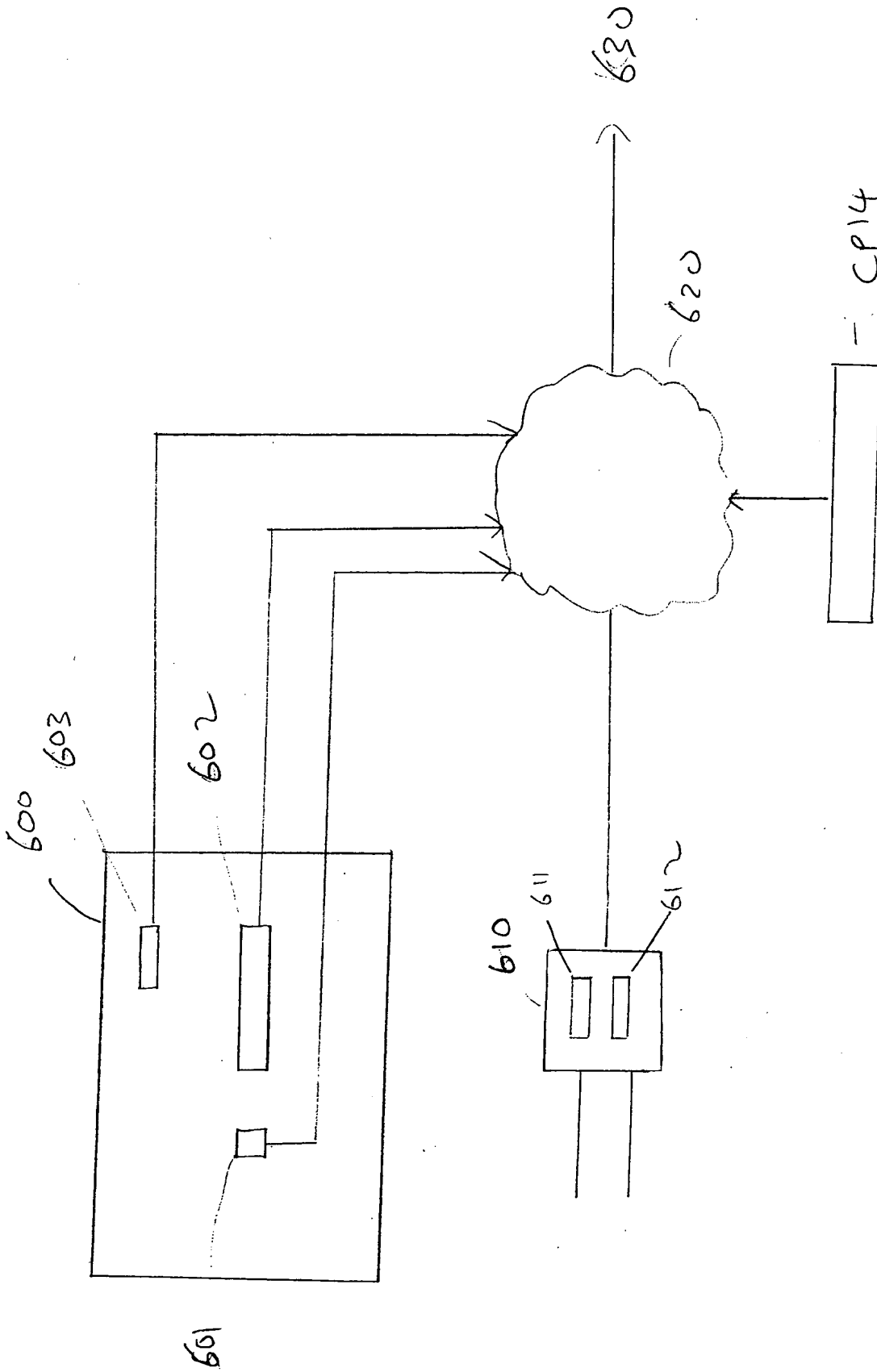


Figure 42

37/39

CP14 bits in Debug and Status Control register			meaning
Secure debug enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No intrusive debug in entire secure world is possible. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are ignored in entire secure world.
1	0	X	Debug in entire secure world is possible
1	1	0	Debug in secure user-mode only. Any debug request, breakpoints, watchpoints, and other mechanism to enter debug state are taken into account in user mode only. (Breakpoints and watchpoints linked or not to a thread ID are taken into account). Access in debug is restricted to what secure user can have access to.
1	1	1	Debug is possible only in some particular threads. In that case only thread-aware breakpoints and watchpoints linked to a thread ID are taken into account to enter debug state. Each thread can moreover debug its own code, and only its own code.

Figure 43A

CP14 bits in Debug and Status Control register			meaning
Secure trace enable bit	Secure user-mode debug enable bit	Secure thread-aware debug enable bit	
0	X	X	No observable debug in entire secure world is possible. Trace module (ETM) must not trace internal core activity.
1	0	X	Trace in entire secure world is possible
1	1	0	Trace is possible when the core is in secure user-mode only.
1	1	1	Trace is possible only when the core is executing some particular threads in secure user mode. Particular hardware must be dedicated for this, or re-use breakpoint register pair: Context ID match must enable trace instead of entering debug state.

Figure 43B

38/39

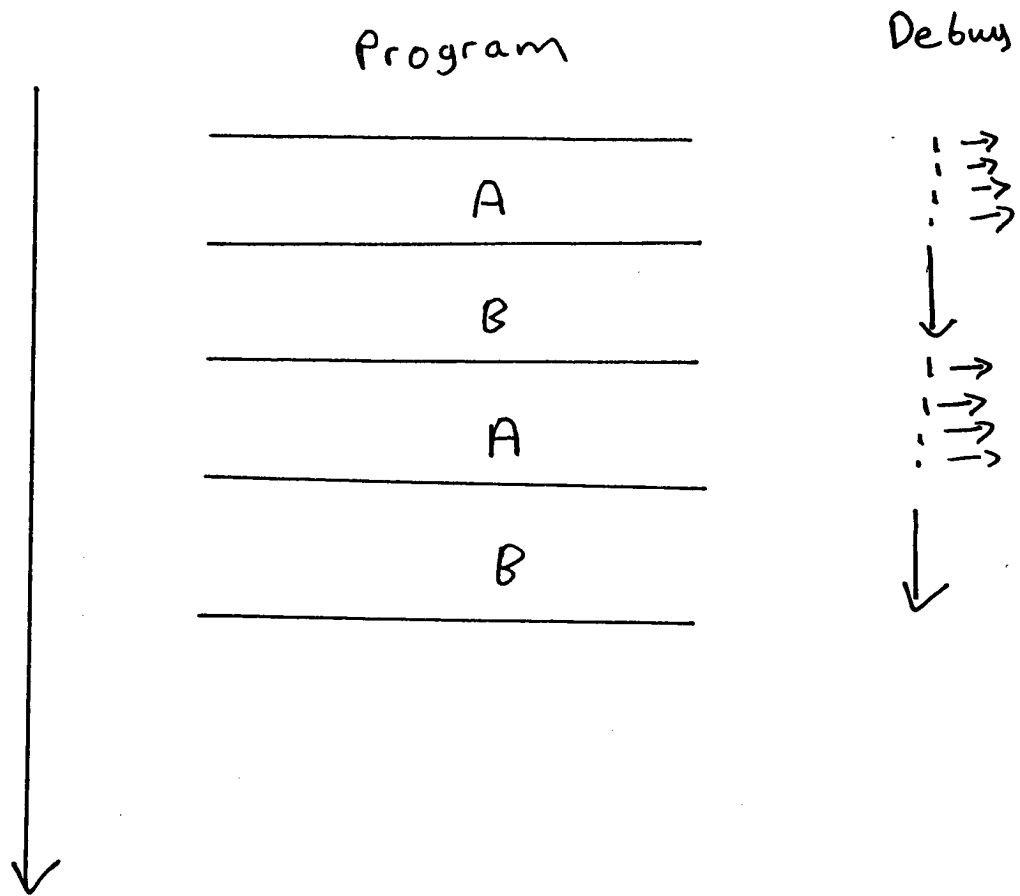


Figure 44

Method of entry	Entry when in non-secure world	entry when in secure world
Breakpoint hits	Non-secure prefetch abort handler	secure prefetch abort handler
Software breakpoint instruction	Non-secure prefetch abort handler	secure prefetch abort handler
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For other non-secure exceptions, prefetch abort.	Disabled for secure data abort and secure prefetch abort exceptions <sup>(1)</sup> . For other exceptions, secure prefetch abort.
Watchpoint hits	Non-secure data abort handler	secure data abort handler
Internal debug request	Debug state in halt mode	debug state in halt mode
External debug request	Debug state in halt mode	debug state in halt mode

<sup>(1)</sup> See information on vector trap register.

(2) Note that when external or internal debug request is asserted, the core enters halt mode and not monitor mode.

**Figure 45A**

Method of entry	Entry in non-secure world	entry in secure world
Breakpoint hits	Non-secure prefetch abort handler	breakpoint ignored
Software breakpoint instruction	Non-secure prefetch abort handler	instruction ignored <sup>(1)</sup>
Vector trap breakpoint	Disabled for non-secure data abort and non-secure prefetch abort interruptions. For others interruption non-secure prefetch abort.	breakpoint ignored
Watchpoint hits	Non-secure data abort handler	watchpoint ignored
Internal debug request	Debug state in halt mode	request ignored
External debug request	Debug state in halt mode	request ignored
Debug re-entry from system speed access	not applicable	not applicable

<sup>(1)</sup> As substitution of BKPT instruction in secure world from non-secure world is not possible, non-secure abort must handle the violation.

**Figure 45B**